
# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**EVENT-DRIVEN SIMULATION AND ANALYSIS OF AN UNDERWATER ACOUSTIC LOCAL AREA NETWORK**

by

Goh, Meng Chong

June 2010

Thesis Advisor: Joseph A. Rice

**Approved for public release; distribution is unlimited**

THIS PAGE INTENTIONALLY LEFT BLANK

| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 |
|---|---|---|---|
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503. | | | |
| **1. AGENCY USE ONLY** ***(Leave blank)*** | **2. REPORT DATE** June 2010 | **3. REPORT TYPE AND DATES COVERED** Master's Thesis | |
| **4. TITLE AND SUBTITLE** Event-Driven Simulation and Analysis of an Underwater Acoustic Local Area Network | | **5. FUNDING NUMBERS** | |
| **6. AUTHOR(S)** Goh, Meng Chong | | | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | | **8. PERFORMING ORGANIZATION REPORT NUMBER** | |
| **9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)** N/A | | **10. SPONSORING/MONITORING AGENCY REPORT NUMBER** | |
| **11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number _____. | | | |
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release; distribution is unlimited | | **12b. DISTRIBUTION CODE** A | |


**13. ABSTRACT (maximum 200 words)**

"Seastar" is an underwater Local Area Network (LAN) concept involving high-frequency acoustic modems sending large volumes of data from a distributed set of peripheral sensor nodes to a centralized node for data fusion. The notional operating range of the acoustic modems used is 500m. This research considers four candidate Seastar network protocols: Time division multiple access (TDMA) polling with and without Selective automatic repeat request (SRQ), and TDMA token ring with and without SRQ. The number of peripheral nodes, the layout of the nodes, and the noise level of the environment are modeled and evaluated, according to performance metrics, including data throughput, communications latency, and packet error rate. It was determined that in a low-noise environment, the token ring with SRQ protocol delivers the most throughput of data with the minimum number of dropped packets, while in high-noise conditions, polling with SRQ is preferred. In addition, if data throughput is not a priority, polling with SRQ is advantageous. Therefore, it is recommended that a switch be implemented for adaptively selecting the network protocol depending on the prevailing noise conditions and the critical performance metrics.


| **14. SUBJECT TERMS** Communications network, sensor network, acoustic communications, undersea sensors, simulation, Seastar, Seaweb | | | **15. NUMBER OF PAGES** 109 |
|---|---|---|---|
| | | | **16. PRICE CODE** |
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UU |

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release; distribution is unlimited**

# EVENT-DRIVEN SIMULATION AND ANALYSIS OF AN UNDERWATER ACOUSTIC LOCAL AREA NETWORK

Goh, Meng Chong
Senior Engineer, Republic of Singapore Navy
B.S. (Honors) in Technology with Electronics, University of London, 2003

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN ENGINEERING ACOUSTICS**

from the

**NAVAL POSTGRADUATE SCHOOL**
**June 2010**

Author: Goh, Meng Chong

Approved by: Joseph A. Rice
Thesis Advisor

Daphne Kapolka
Chair, Engineering Acoustics Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

"Seastar" is an underwater Local Area Network (LAN) concept involving high-frequency acoustic modems sending large volumes of data from a distributed set of peripheral sensor nodes to a centralized node for data fusion. The notional operating range of the acoustic modems used is 500m. This research considers four candidate Seastar network protocols: Time division multiple access (TDMA) polling with and without Selective automatic repeat request (SRQ), and TDMA token ring with and without SRQ. The number of peripheral nodes, the layout of the nodes, and the noise level of the environment are modeled and evaluated, according to performance metrics, including data throughput, communications latency, and packet error rate. It was determined that in a low-noise environment, the token ring with SRQ protocol delivers the most throughput of data with the minimum number of dropped packets, while in high-noise conditions, polling with SRQ is preferred. In addition, if data throughput is not a priority, polling with SRQ is advantageous. Therefore, it is recommended that a switch be implemented for adaptively selecting the network protocol depending on the prevailing noise conditions and the critical performance metrics.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS

# LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| BER | Bit Error Rate |
| BPP | Bits Per Packet |
| $\varepsilon_b$ | Energy Per Bit |
| LAN | Local Area Network |
| MSKF | M-ary Shift Key |
| $N_0$ | Noise Energy Level |
| NL | Noise Level |
| NSL | Noise Spectrum Level |
| PER | Packet Error Rate |
| PSK | Phase Shift Key |
| SL | Source Level |
| SNR | Signal-to-Noise Ratio |
| $SNR_b$ | Signal-to-Noise Ratio Per Bit |
| SRQ | Selective Automatic Repeat Request |
| TL | Transmission Loss |
| WAN | Wide Area Network |

THIS PAGE INTENTIONALLY LEFT BLANK

# ACKNOWLEDGMENTS

I would like to thank the following people for making my learning experience at the Naval Postgraduate School (NPS) a joy, and this period of my life a truly memorable one:

First and foremost, to my dearest wife, Sarah, for her love, patience, support and encouragement. Thank you for bearing our first child, Zoey, and taking care of her. To my child, Zoey, you are a joy to behold and hold onto.

To my parents, for inculcating in me the discipline to persevere in life and for coming to Monterey to provide care and support when Zoey was born.

To my thesis advisor, Joe Rice, for giving me this opportunity to be involved in the developmental work of Seastar. Thank you for providing invaluable guidance, patience and freedom for the successful completion of this thesis.

To Chris Fletcher from SPAWAR, thank you for sharing your knowledge of the deployment and configurations of the Seastar/Seaweb network.

To the staff from Teledyne Benthos, Ken Scussel and Kevin Amundsen, for answering my questions about the Seastar modem testing and specification.

To the faculty and staff of the Physics Department at NPS, who helped me to garner a suitable basis of knowledge for completing this thesis. In particular, I would like to acknowledge Professors Daphne Kapolka, Kevin Smith, Steve Baker, and Bruce Denardo for their help in my more advanced acoustics courses.

To LT Devlin Messmer and family, thank you for volunteering as my sponsor, helping my family transition to living in the USA, and extending friendship toward my family and myself.

Last, but not least, I would like to thank my colleagues and fellow students, LT Jeremy Biediger, LT Pongsakorn Sommai (Royal Thai Navy) and ENS William Jenkins, who provided peer advice and moral support, without which I could not have finished my thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

Exploiting the underwater domain has been a challenge since the onset of the First World War, when the modern submarine was introduced to the world. Since then, technological innovation has enabled surveillance of the undersea environment. From the mid 20th century, wired acoustic sensors have been deployed on the seabed for monitoring the radiated signatures of submarines. These sensors, however, are expensive to deploy and maintain. With the transition into the network-centric warfare of the 21st century and increasing interest in littoral anti-submarine warfare (ASW), there is a demand for a cheaper, more rapidly deployable system to provide underwater domain awareness.

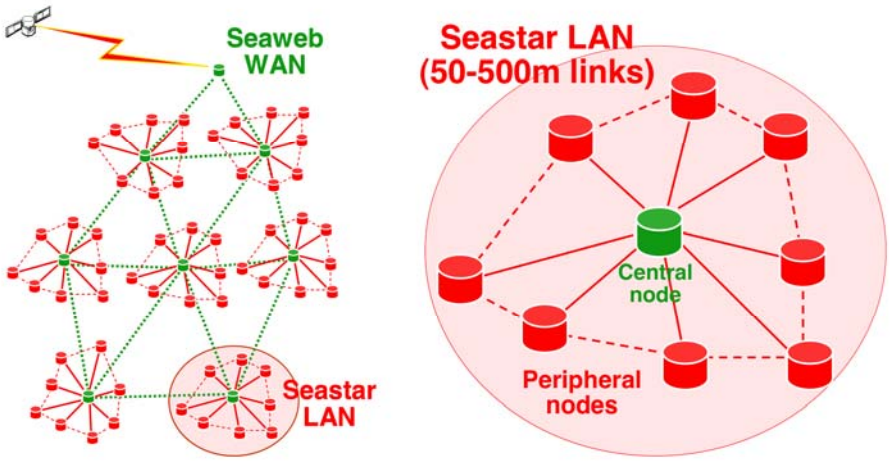


Figure 1. Seastar LAN within a Seaweb WAN. (After [1])

## A. SEASTAR NETWORK

The Naval Postgraduate School (NPS), with sponsorship from the Office of Naval Research, is developing the Seaweb Wide Area Network (WAN), which incorporates underwater acoustic modem technology. Seaweb allows a network of autonomous battery-powered sensors to be rapidly deployed over wide areas (10–100km$^2$). To increase the density of sensors for a smaller area, a study was done by [1] in 2007 to consider feasibility of a local area network (LAN) consisting of short-range acoustic

modems capable of communicating at distances up to 500m. With its higher operating frequency and greater spectral bandwidth, the acoustic LAN would carry large quantities of data from the peripheral nodes for fusion at a central node. As shown in Figure 1, the fused information may then be transmitted from the Seastar central node via the Seaweb WAN for further uses. The acoustic LAN is named "Seastar" in reference to its centralized hub and spoke architecture.

## B. SCOPE OF RESEARCH

The scope of this thesis research is to simulate candidate Seastar networking protocols and evaluate their operations against a set of performance metrics. This simulation involves representing the high-frequency acoustic modem, the spectral bandwidth, the operational range, and the associated data rate in an event-driven numerical program.

The performance metrics include information throughput (bytes), latency (seconds), and dropped data rate (data packets per day) of the network compared to the number of nodes the network can support, layout of the network, and the ambient environmental noise level.

## C. THESIS ORGANIZATION

The thesis contains seven chapters including this introduction.

Chapter II gives an overview of the Seastar prototype short-range modem and its programmable functions.

Chapter III describes the physical ocean operating environment, including the sonar equation and a link budget model for underwater acoustic communication.

Chapter IV discusses the network protocols implemented in the simulation.

Chapter V presents the simulation setup, establishes a set of performance metrics, and identifies various test cases to be simulated.

Chapter VI reports the results of the simulation test cases.

Chapter VII draws conclusions and discusses recommendations for future work.

# II. HIGH–FREQUENCY ACOUSTIC TELEMETRY MODEM

The concept for the Seastar network and the specification of the short-range acoustic modem were introduced by Kerstens [1]. As a result of that study, a high-frequency modem was developed at the request of NPS to provide an operational frequency band of 35 to 55 kHz for the implementation of the Seastar network.

The high–frequency acoustic telemetry modem is an evolution of the ATM-88X family of acoustic telemetry modems that have been used in the implementation of Seaweb. These modems provide wireless bidirectional underwater communications between two or more nodes, as illustrated in Figure 2.

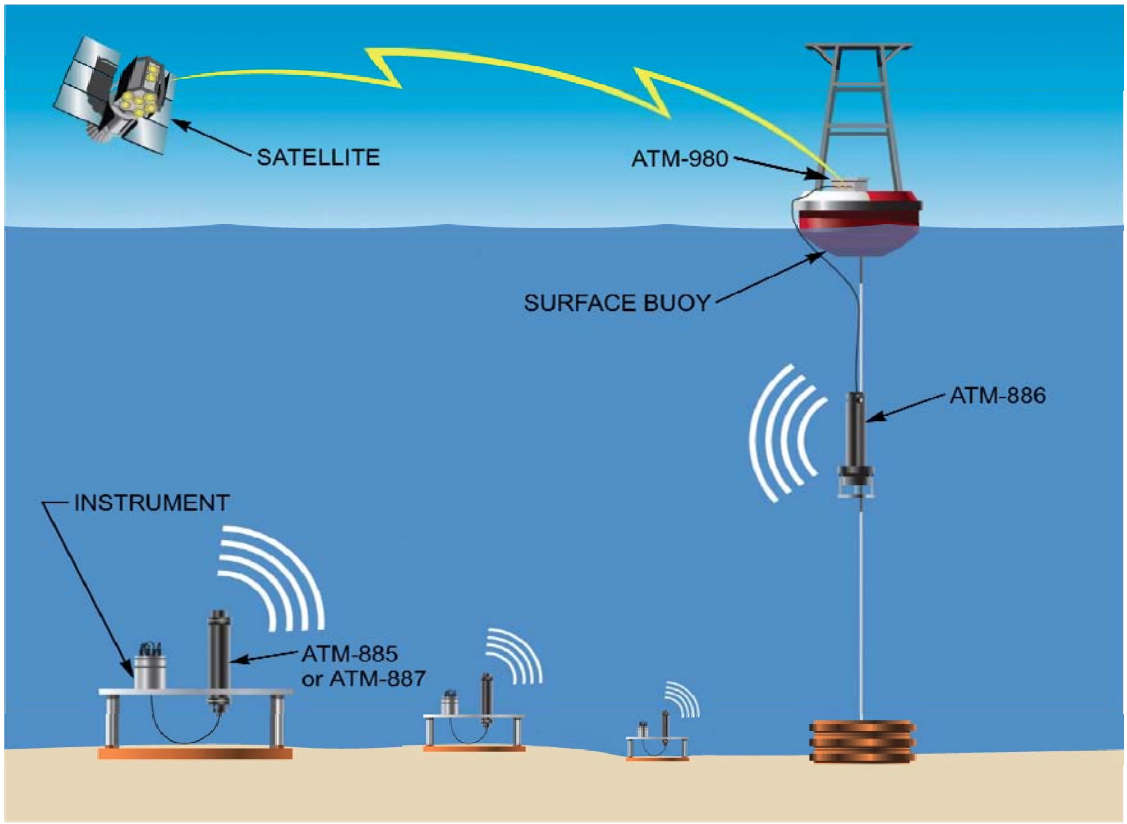


Figure 2. Family of acoustic telemetry modems. (From [2])

## A. SPECIFICATION

The general specification of the high–frequency acoustic telemetry modems are presented in Table 1.

| Type | Seaweb Modem | Seastar Modem |
|---|---|---|
| Frequency Band | 9–14 kHz (LF omnidirectional) | 35–55 kHz (HF omnidirectional) |
| Data Modulation | PSK and Multi-Channel MFSK | |
| Bit Rate for Multi-Channel MFSK | 140 – 2400 bits/s | 560 – 9600 bits/s |
| Processing Feature | Data Redundancy<br>½-rate Convolutional Coding<br>Multipath Guard Period | |
| Average Transmit Power | 20 watts at power level 08 (max) @ 21 VDC<br>Source Level: 183 dB re: 1 µPa @ 1 m | |
| Battery Capacity | 400 watt-hours at max power level 08 @ 21°C | |
| Range | 500 – 5000 m | 50 – 500 m |

Table 1. General specifications of ATM-890 acoustic telemetry modems.

### 1. Frequency

The Seastar omnidirectional transducer operates in a 20 kHz bandwidth of relatively flat frequency response from between 35 kHz and 55 kHz. This is a four-fold increase over Seaweb's spectral bandwidth, which operates in the 9–14 kHz range. This should result in a higher throughput of data for the Seastar network at the expense of a shorter range of transmission.

## 2. Transmission Mode

There are two modulation schemes available aboard the modem: phase shift keying (PSK) and multi-channel M-ary frequency shift keying (MFSK).

PSK modulation permits the modem to transmit at a higher bit rate than multi-channel MFSK. However, as sound propagates, the ocean environment produces a multi-path effect, which greatly reduces the coherent nature of the PSK signal. Thus, PSK modulation is reserved only for transmission in the vertical column of the ocean.

For the purpose of the Seastar network, we would only operate the modem using the multi-channel MFSK modulation scheme, which ensures a more robust physical layer for the network. With multi-channel MFSK, 32 individual pulse trains of 4-ary FSK are transmitted simultaneously. Thus, at any instant during data transmission, 32 out of a possible 128 tones are active. Jenkins [3] examines various candidate implementations of multi-channel MSFK modulation for Seastar.

## 3. Data Reliability

There are constraints in underwater communications, such as a slow sound speed, signal fading, frequency-dependent attenuation, limited spectral bandwidth, and multipath interference due to sea surface and sea floor reflections. These channel impairments are treated in greater detail in the next chapter. The modem uses three different methods to overcome channel effects and increase the reliability of the transmission. These methods involve the use of convolutional coding, multipath guard periods, and data redundancy. However, these methods reduce the effective bit rate or throughput of the transmission.

### *a. Convolutional Coding*

Convolutional coding is a form of forward error correction. The modem uses a constraint-length-9, ½-rate convolutional coding. This allows the modem to recover binary information even when a few symbols are lost at the receiver. The effective bit rate or throughput is reduced by half.

### *b. Multipath Guard Period*

In the high multipath situations where there is significant time spread at the receiver, a guard period may be inserted between data symbols. If the duration of the guard period is the same length as the data symbols, the effective bit rate is reduced by half.

### *c. Data Redundancy*

Data redundancy involves repeating the same data bits two or more times. This reduces the bit rate by half or more, depending on the number of times the data bits are repeated.

## 4. Data Rate

The data rate of the modem is selectable from a range of 140 bits/sec to 2400 bits/sec for multi-channel MFSK modulation as described in Table 2. Note that these data rates may be increased by a factor of 2 or 4, depending on how the additional spectral bandwidth at high frequencies is exploited [3].

| Setting | Remarks |
|---|---|
| 02 | 140 bits/s MFSK repeated four times with ½-rate convolutional coding and 25 ms multipath guard period |
| 03 | 300 bits/s MFSK repeated twice with ½-rate convolutional coding and 25 ms multipath guard period |
| 04 | 600 bits/s MFSK with ½-rate convolutional coding and 25 ms multipath guard period |
| 05 | 800 bits/s MFSK with ½-rate convolutional coding and 12.5 ms multipath guard period |
| 06 | 1066 bits/s MFSK with ½-rate convolutional coding and 3.124 ms multipath guard period |
| 07 | 1200 bits/s MFSK with ½-rate convolutional coding |
| 08 | 2400 bits/s MFSK |

Table 2. Bit rate selection for MFSK.

The data rate is affected by the choice of data reliability methods employed. The greater the protection, the higher the data reliability will be, but the lower the data rate. For the purpose of this thesis, we will assume 140 bits/s as the rate for utility packet transmission and 2400 bit/s for data packet transmission.

## B. SPECIAL FUNCTIONS

There are two special modem functions of particular interest to this thesis: acoustic ranging and transmit power.

### 1. Acoustic Ranging

A modem and its sensor packet are generally deployed by dropping the modem overboard from a boat or aircraft and marking the position of the point of release. However, depending on the currents and depth of the area, the position drifts from where it is dropped to where it ultimately anchors on the seabed. Therefore, there is a location uncertainty.

To correct such errors, the modem can be remotely activated to measure the range between itself and neighboring modems in the network, by issuing an acoustic ranging signal as discussed by [14].

### 2. Transmit Power

While high transmit power will ensure a high signal-to-noise power ratio (SNR) that supports maximum transmission ranges, excessive transmit power is not desirable, as it will also contribute to a high reverberation level that reduces the reliability of short-range communication. High transmit power also reduces the life of the node, as it consumes more battery power.

Modem settings allow for a remote power command to set the transmit power. The available settings are as shown in Table 3, with the values indicating the reduction in power from the maximum level.

| Setting | Power Level (dB) |
|---|---|
| 01 | - 21 dB (Minimum power level) |
| 02 | - 18 dB |
| 03 | - 15 dB |
| 04 | - 12 dB |
| 05 | - 09 dB |
| 06 | - 06 dB |
| 07 | - 03 dB |
| 08 | 0 dB (Maximum power level) |

Table 3. Transmit power settings.

## C. OPERATIONAL SETTINGS

This section introduces some of the modem settings used in the network simulations.

### 1. Acoustic Response Time-Out

The acoustic response timeout is the time during which the local modem waits for an acknowledgment of an acoustic command sent to a remote modem. It is computed from the expected round-trip time of the command, acknowledgment dialog, and duration of the transmitted packets. If the wait for an acknowledgment exceeds this timeout, a dropped packet is declared.

### 2. Idle Time

When the modem does not receive input from a source after the idle time period, it goes into a low-power state to conserve energy. Therefore, if the modem is accessed only once a day, the idle time should be set as low as possible to conserve energy. However, if the modem is constantly being polled, the idle time should be set longer to avoid the need to frequently reawaken the modem from its low-power state. The calculation of the latency time as a performance metric will allow us to determine appropriate settings for the idle time.

# III. THE COMMUNICATION CHANNEL

## A. PHYSICAL CHANNEL

Radio transmission in terrestrial networks is a well-studied topic resulting from advances in commercial telecommunication. Acoustic communication in the ocean is far less understood. There are many factors that impair signal reception in the ocean. The air-sea interface and the sea bottom boundary support multipath propagation; the slow rate of acoustic propagation causes signal delays; the spatially and temporally varying sound-speed profile causes convergence and divergence zones in the water column; and various noise sources, together with the frequency and bandwidth of the signal, cause the signal to become delayed, distorted and weakened.

In the following sections, we categorize these factors that influence acoustic communication into a few parameters, i.e., transmission loss, ambient noise level, and source transmission level. By focusing on the frequency bandwidth and the operational range of the Seastar modem, we will apply these parameters in a link budget analysis to deduce the SNR at the receiver. From this SNR, we hope to estimate the probability of error for utility and data packets to be used in our simulation.

## B. TRANSMISSION LOSS

Transmission Loss (TL) reflects the fraction of sound intensity lost between the source and the receiver. These losses can be broadly categorized as geometric spreading loss and the absorption losses. Because the range of these high frequency modems is short, we only consider simple spreading and absorption estimates.

### 1. Spreading Loss

Let $I_0$ be the reference intensity of the sound pressure at a range of 1 m from the source and $I_1$ be the sound intensity at a distant point. The transmission loss is

$$TL = 10 \log \frac{I_0}{I_1} \quad \text{(dB)} \tag{3.1}$$

### *a. Spherical Spreading*

For a small source in a homogeneous, unbounded and lossless medium, power *P* generated at the source will radiate spherically outward.

$$P = 4\pi r_0^2 I_0 = 4\pi r_1^2 I_1 \tag{3.2}$$

If $r_0$ is a reference distance of 1 m, the transmission loss at a distance $r_1$ will be

$$TL_{spherical} = 10\log\frac{I_0}{I_1} = 10\log r_1^2 = 20\log r_1 \text{ dB} \tag{3.3}$$

where $r_1$ is in meters.

### *b. Cylindrical Spreading*

The ocean medium is bounded by the ocean surface and the seabed. In an ideal waveguide, sound does not cross these two boundaries. Therefore, the power generated at the source will radiate cylindrically outward, bounded between two parallel planes separated by a depth of *D* meters.

$$P = 2\pi r_0 D I_0 = 2\pi r_1 D I_1 \tag{3.4}$$

$$TL_{cylindrical} = 10\log\frac{I_0}{I_1} = 10\log r_1 \text{ (dB)} \tag{3.5}$$

## 2. Absorption Loss

In general, absorption terms in TL calculations arise from scattering, as well as from thermal and chemical relaxations and are frequency dependent. In this discussion scattering is assumed to be negligible. In this case absorption loss is dominated by thermal and chemical relaxations in seawater. An estimate of the absorption coefficient $\alpha$ is defined as follows by Thorp [4] as

$$\alpha = \frac{0.11 f^2}{1+f^2} + \frac{44 f^2}{4100+f^2} + 3\times 10^{-4} f^2 + 3.3\times 10^{-3} \text{ (dB/km)} \tag{3.6}$$

$$TL_{absorption} = \alpha r \times 10^{-3} \text{ (dB)} \qquad (3.7)$$

where $r$ is in meters and $f$ is in kHz.

$TL_{absorption}$ is shown in Table 4 for a frequency range from 35 to 55 kHz at a range of 500 meters.

| Frequency | $TL_{absorption}$ (dB) |
|---|---|
| 35 kHz | 5.30 |
| 40 kHz | 6.47 |
| 45 kHz | 7.63 |
| 50 kHz | 8.76 |
| 55 kHz | 9.85 |

Table 4. TL due to absorption at a range of 500 m.

From Table 4, we notice that the maximum TL will occur at a frequency of 55 kHz. Since the modem makes use of the full frequency spectrum, we will use the $TL_{absorption}$ factor at 55 kHz to compute the total transmission loss. To compute the maximum TL, spherical spreading is used. Therefore, by using Equations (3.3) and (3.7),

$$TL = 20\log(500) + \alpha 500 \times 10^{-3} = 53.98 + 9.85 = 63.83 \text{ (dB)} \qquad (3.8)$$

## C. NOISE LEVEL

Noise Level (NL) in the ocean can be categorized into man-made noise and ambient noise. Man-made noise is mainly caused by machinery noise along the coast and shipping activity, while ambient noise is mainly caused by biologic and seismic activity, and by hydrodynamic noise caused by wave action, currents, tides, wind, and rain.

Coates [5] provides empirical formulae to estimate Noise Spectrum Levels (NSL) as a function of frequency for open water. It can be observed from Figure 3 that in the band of our frequency of interest, between 35 and 55 kHz, wind-related noise contributes the most to the NSL.

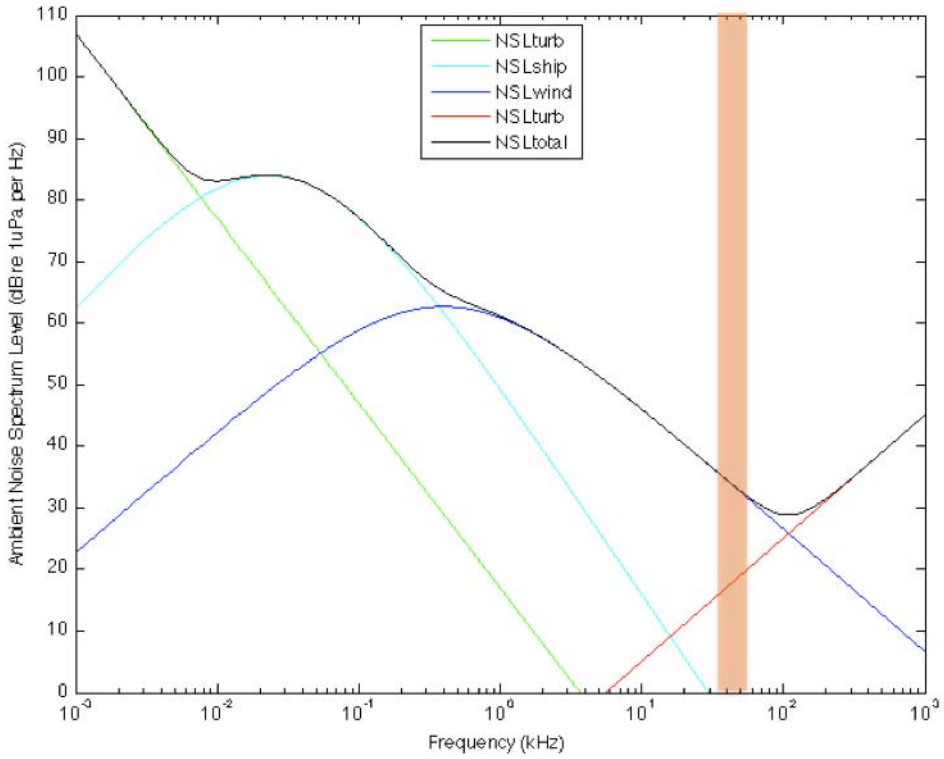


Figure 3. Noise Spectrum Level based on empirical formulae by Coates. (After [5])

Our interest here is in the littoral waters where sensors are likely to be deployed. The noise level in littoral water arises from three main components, as stated by Urick [6]. They are: (1) wind noise, (2) shipping and industrial noise, and (3) biological noise. A power sum of the noise spectrum level from these sources is used to determine the noise level of the area. However, as such noise varies over time, only a rough estimate may be made.

### 1. Wind and Sea State

According to the empirical formula from [5], illustrated in Figure 4, at a frequency of 45 kHz and for wind speeds between 0 and 15 m/s, the NSL ranges from 20 to 45 dB re: 1 $\mu Pa^2/Hz$. This is comparable to a wind contribution ranging from 38 to 40 dB with wind speeds ranging from between 6 m/s and 10 m/s, as shown in Figure 5.

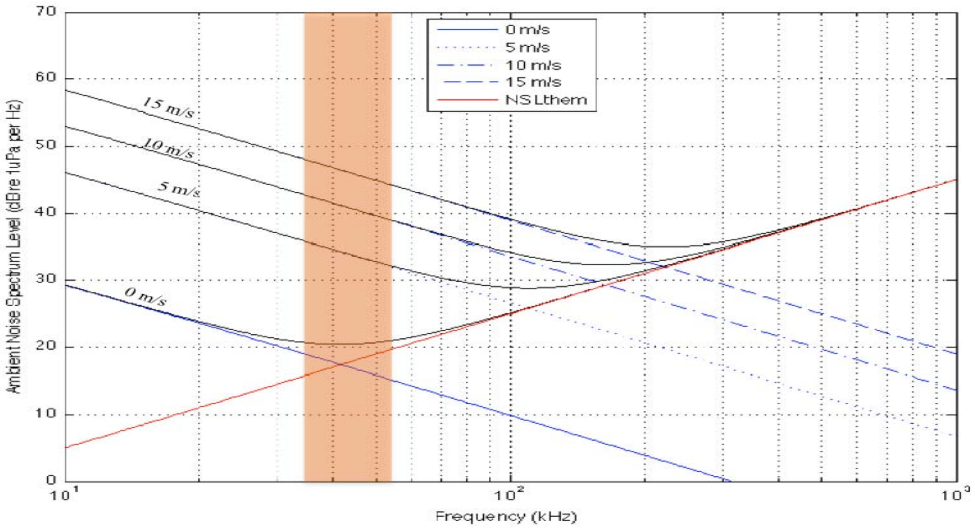


Figure 4. Ambient Noise Spectrum Level caused by sea state effect. (After [5])

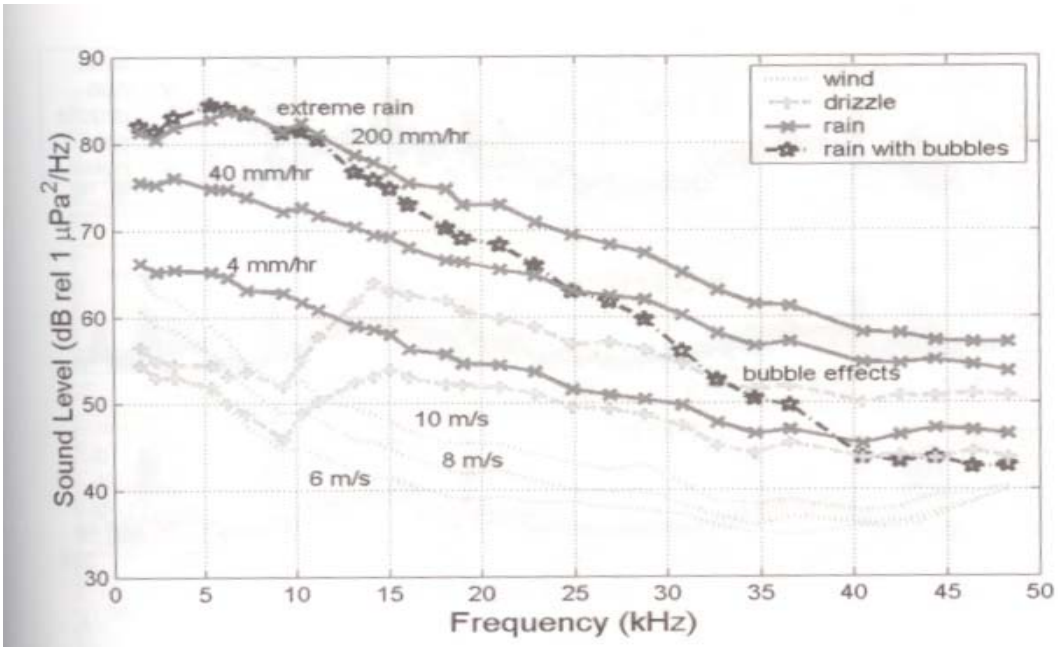


Figure 5. Noise Spectrum Levels of wind and rain. (From [10])

## 2. Shipping – In Bays and Harbors

Most of the shipping noise in the deep ocean is below 1 kHz. Even in coastal areas, shipping noise contributes little to our frequency band of interest. Anderson and Gruber [7] measured the ambient noise at 30, 90, and 150 kHz in five ports in the United

States and Panama. The highest NSL value measured was from the port of Cristobal, where 70 dB at 30 kHz and 65 dB at 90 kHz were recorded. The lowest noise level was from Norfolk, where 45 dB at 30 kHz and 44 dB at 90 kHz were recorded. Extrapolations were made from these two measurements to obtain the NSL for 45 kHz, which has a maximum NSL of 77 dB re: 1 $\mu Pa^2$/Hz and a minimum NSL of 44 dB re: 1 $\mu Pa^2$/Hz.

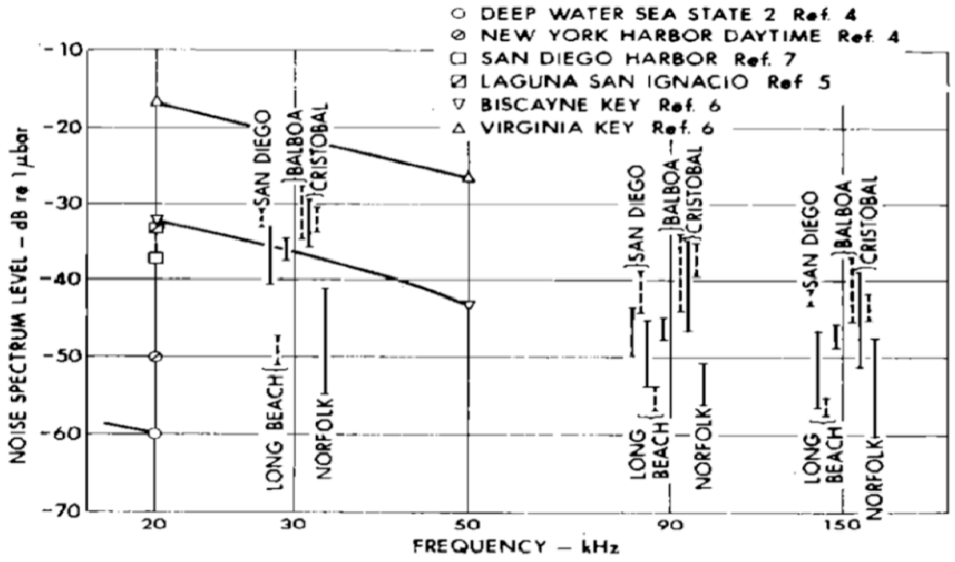


Figure 6. Noise spectrum data obtained from five different ports in the United States and Panama. The dotted line represents data taken at night. Note also that the vertical scale is dB re 1 µBar instead of re 1 µPa. (From [7])

## 3. Biology

Snapping shrimp, which exist in waters below latitudes of 27 degrees, generate a high level of noise in the frequency band of our interest. From Figure 7, we see that at a frequency of 45 kHz, snapping shrimp can generate NSL between 48 and 74 dB re 1 $\mu P^2$/Hz.

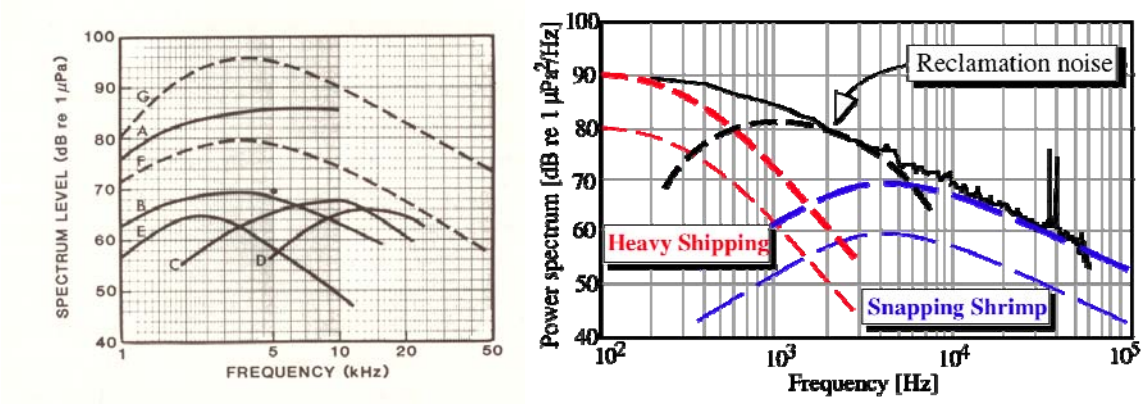


Figure 7. Left: Snapping shrimp NSL measured at various locations. (From [8]). Right: NSL measured in waters off the coast of Singapore. (From [9])

Therefore, the total noise spectrum level (NSL) of the ambient noise is the summation of the three major sources of noise: shipping, wind and biologic noise. Since the range of a noise source is variable, such as the presence or absence of snapping shrimp, noise spectrum level is also variable. We therefore determine some minimum and maximum values for the total NSL:

$$NSL_{\min} = Shipping_{\min} \oplus Bio_{\min} \oplus Wind_{\min}$$

$$NSL_{\min} = 44 \text{ dB} \oplus 0 \text{ dB} \oplus 0 \text{ db} = 44 \text{ (dB)} \tag{3.9}$$

$$NSL_{\max} = Shipping_{\max} \oplus Bio_{\max} \oplus Wind_{\max}$$

$$NSL_{\max} = 70 \text{ dB} \oplus 74 \text{ dB} \oplus 45 \text{ db} = 75.5 \text{ (dB)} \tag{3.10}$$

To translate from NSL to NL, we must account for the transmission bandwidth, which we assume to be 80 Hz for each multi-channel MFSK tone.

Therefore,

$$NL_{\min} = NSL_{\min} + 10\log(80) = 44 + 19 = 63 \text{ (dB)} \tag{3.11}$$

$$NL_{\max} = NSL_{\max} + 10\log(80) = 75.5 + 19 = 94.5 \text{ (dB)} \tag{3.12}$$

## D. LINK BUDGET

For an omnidirectional transmitter, the source level (SL) is defined as

$$SL = \frac{I_{souce@1m}}{I_{ref}} \tag{3.13}$$

The intensity of the source is referenced to a standard range of 1 m. At a radius of 1 m, the acoustic source is surrounded by a sphere of surface area

$$Area_{sphere} = 4\pi r^2 = 12.6 \text{ m}^2 \tag{3.14}$$

The power of the Seastar modem is 20 watts. Therefore, the intensity of the source is

$$I_{scource@1m} = \frac{20}{12.6} = 1.59 \text{ W/m}^2 \tag{3.15}$$

The data rate for the utility packet and data packet is 160 bps and 2400 bps, respectively. Therefore, the intensity of the source or the energy-per-bit is as follows:

$$I_{util/bit} = \frac{I_{source@1m}}{DataRate_{util}} = \frac{1.59}{160} = 9.92 \times 10^{-3} \text{ W/m}^2 \tag{3.16}$$

$$I_{Data/bit} = \frac{I_{source@1m}}{DataRate_{Data}} = \frac{1.59}{2400} = 6.61 \times 10^{-4} \text{ W/m}^2 \tag{3.17}$$

The intensity of the water reference at 1 $\mu$Pa is $6.76 \times 10^{-19}$ W/m$^2$ [9]. Therefore, by applying Equations (3.16) and (3.17) respectively, together with the reference intensity of water to (3.13), the SL or energy-per-bit for utility packets and data packets ($\varepsilon_{b\_Util}$ and $\varepsilon_{b\_Data}$) is:

$$\varepsilon_{b\_Util} = 10\log(\frac{I_{util/bit}}{I_{ref}}) = 161.7 \text{ dB} \tag{3.18}$$

$$\varepsilon_{b\_Data} = 10\log(\frac{I_{Data/bit}}{I_{ref}}) = 149.9 \text{ dB} \tag{3.19}$$

The SNR-per-bit ($SNR_b$) can be found by using the Link Budget Model developed by Hanson [12] in the following formula:

$$\text{Energy Per Bit - Transmission Loss - Noise Level } = SNR_b \tag{3.20}$$

By substituting Equations (3.18) and (3.19) for Source Level, Equation (3.8) for Transmission Loss, and Equation (3.11) for Noise Level to find the maximum $SNR_b$, and Equation (3.12) for Noise Level maximum to find the minimum $SNR_b$, we get the following result:

$$SNR_{Util\_Max} = \varepsilon_{b\_Util} - TL - NL_{\min} = 34.9 \text{ dB} \tag{3.21}$$

$$SNR_{Util\_Min} = \varepsilon_{b\_Util} - TL - NL_{\max} = 3.4 \text{ dB} \tag{3.22}$$

$$SNR_{Data\_Max} = \varepsilon_{b\_Data} - TL - NL_{\min} = 23.1 \text{ dB} \tag{3.23}$$

$$SNR_{Data\_Min} = \varepsilon_{b\_Data} - TL - NL_{\max} = -8.4 \text{ dB} \tag{3.24}$$

We must be aware that the above $SNR_b$ may not be the minimum $SNR_b$ as the NL is given as an average over time. At the other extreme, a noisy source (such as a speed boat) passing above the receiver will generate a greater NL at that instant which will affect the $SNR_b$. Therefore, high variability and large dynamic range can be expected at the receiver input.

Another observation is that the $SNR_b$ for a utility packet is about 12 dB stronger than the $SNR_b$ for the data packet, because the utility packet has a slower rate of transmission.

## E. ERROR RATE

There are two selectable bandwidths for the Seastar modem, 10 kHz and 20 kHz. The band is divided into 32 sets of 4-frequency bands. This is known as a multi-channel 4-ary FSK modulation. [3] describes alternative implementations of multi-channel MFSK for Seastar, but these are not considered in this thesis.

The formula in [11] is used to compute a binary Bit Error Rate ($BER_b$) for a Gaussian noise random variable as

$$BER_b = 0.5 * erfc(\sqrt{\frac{\varepsilon_b}{N_0}}), \tag{3.25}$$

where $erfc(\bullet)$ is the complementary error function, $\varepsilon_b$ is the signal energy level per bit, and $N_0$ is the noise energy level per bit. $\frac{\varepsilon_b}{N_0}$ is equivalent to the SNR of the transmitted signal. Therefore Equation (2.9) is rewritten as

$$BER_b = 0.5 * erfc(\sqrt{SNR_b}). \tag{3.26}$$

For a M-ary FSK modulation, the probability of signal error, from [10], is

$$P_M = \sum_{n=1}^{M-1} (-1)^{n+1} \binom{M-1}{n} \left(\frac{1}{n+1}\right) \exp\left[-\frac{nk\varepsilon_b}{(n+1)N_0}\right], \tag{3.27}$$

where $k$ is the number of encoded bits in M (or $k = \log_2(M)$).

The BER for M-ary FSK can be related to the probability of signal error using the following relationship:

$$BER = \frac{2^{k-1}}{2^k - 1} P_M. \tag{3.28}$$

There are 9 bytes in a utility packet and up to 4096 bytes per data packet. Therefore, the number of bits per packet for a utility packet ($BPP_{Util}$) and a data packet ($BPP_{Data}$) are as follows:

$$BPP_{Util} = 9 \times 8 = 72 \text{ bits} \tag{3.29}$$

$$BPP_{Data} = 4096 \times 8 = 32768 \text{ bits} \tag{3.30}$$

The packet error rate (PER) for both utility and data packets can be found by using Equations (3.28), (3.29) and (3.30) as shown below:

$$PER_{Util} = 1 - (1 - BER)\hat{}(BPP_{Util}) \tag{3.31}$$

$$PER_{Data} = 1 - (1 - BER)\hat{}(BPP_{Data}) \tag{3.32}$$

*PER* for both utility and data packets are shown in Figure 8.

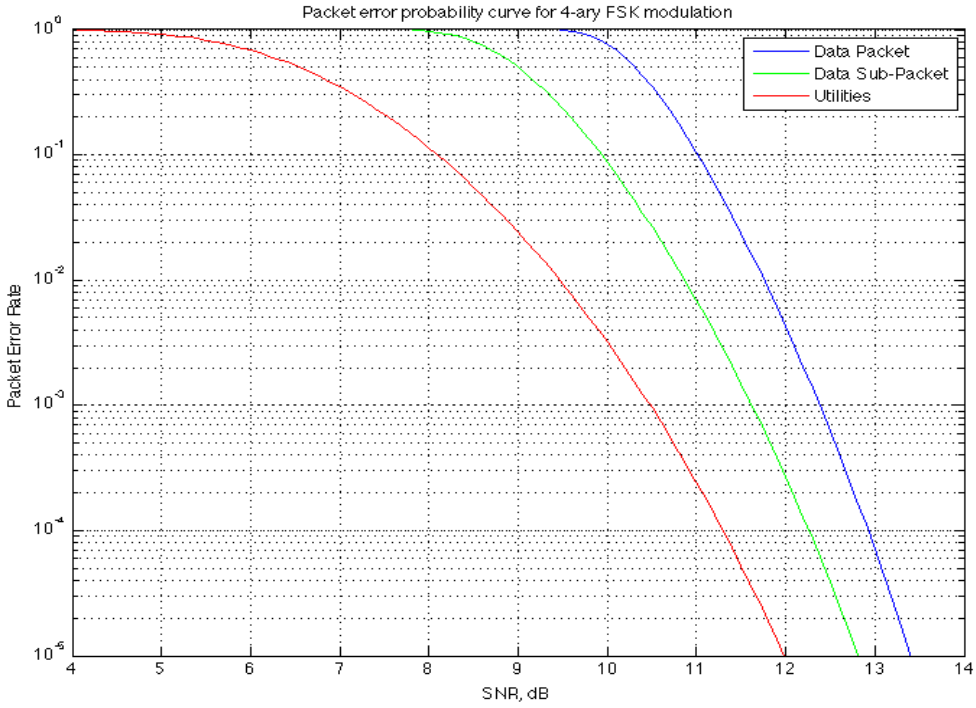


Figure 8. Packet Error Rate (PER) for utility and data packets.

From Figure 8, we observe that the *PER* for data packets is about 400 times greater than for utility packets. This factor is attributable to the larger number of bytes (4096 verse 72) in a data packet compared to a utility packet and ignores the receiver processing gain benefitting the utility packet by virtue of its lower bit rate. If the $SNR_b$ difference due to the different bit rate for data and utility packets is taken into account, the *PER* for the utility packet curve in Figure 8 would be an additional 12 dB less than that of the data packet *PER*, rendering the *PER* for the utility packets negligible when compared to the *PER* for data packets. We neglect the $SNR_b$ difference in our simulation.

Our assumed dependence of the communication channel link on the noise level is summarized as follows. For a given transmit power and communication distance, we have a constant value in $\mathcal{E}_b$ and TL, and SNRb varies solely from fluctuation in the noise level. This range of variation in SNRb determines the range of probabilities of packet error rate according to Figure 8. Therefore, if we can obtain an accurate measurement of the noise level in the communication channel, we can calculate SNR and estimate the

error rate of the channel. For our simulation, we vary the data PER from 0 to 0.8 to observe the effect of noise on the network protocol. In these simulations, we hold the value of the utility PER to be 400 times lower than the data PER.

# IV. NETWORK PROTOCOLS

Seaweb uses a tree topology, which supports routing of information from the branch nodes to a gateway node. In Seastar, where the central node acts as a direct-path data collection point from a number of peripheral nodes, the tree topology is limited to a single link per branch, often described as a “star” topology. Previously, Kerstens [1] has identified the polling and token ring protocols to be suitable for implementation of Seastar.

This chapter describes the polling and token ring protocols as well as improvements of these two by implementing a selective automatic repeat request (SRQ). A basic network exchange is described in Figure 9.

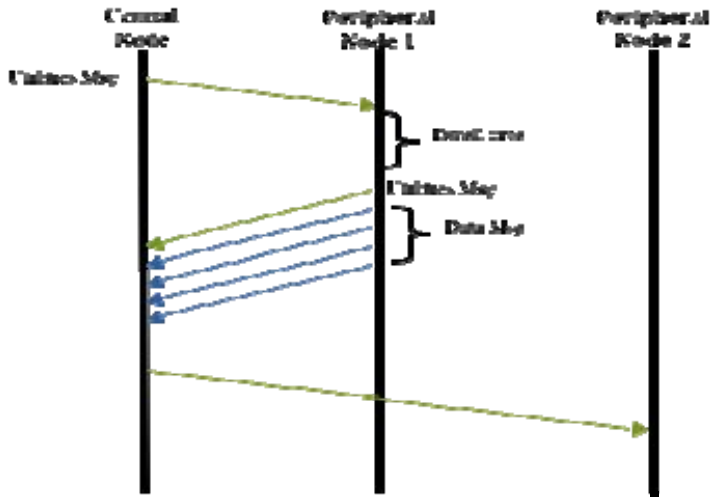

Figure 9. Network exchange consists of a utility packet sent from the central node to the peripheral node polling for data. After a short delay, the peripheral node replies with a utility packet concatenated with a data packet. The data packet is broken into 16 sub-packets.

Seastar communications involve two classes of packets. The first class provides various utility functions for the network and is referred to as a “utility packet.” The utility packet is transmitted at a low bit rate (due to convolutional coding and data redundancy) to ensure data reliability.

The second class of packet is the “data packet.” The data packet is transmitted at a higher bit rate.

## A. SELECTIVE AUTOMATIC REPEAT REQUEST (SRQ)

The SRQ, as described in [13], is a mechanism implemented at the link layer to mitigate the unreliability inherent in the acoustic modem physical layer. Without SRQ, when the central node receives a data packet that contains bit errors, it will either drop that data packet or request that the whole data packet be retransmitted again. The channel capacity is thus reduced.

SRQ improves the link reliability and channel capacity by splitting each data packet into 16 sub-packets, each with a 2-byte cyclic redundancy check. When errors occur in one or more of the sub-packets, the central node requests that only the corrupted sub-packets be retransmitted, as shown in Figure 10. This reduces the number of data bytes to be transmitted by as much as a factor of 16. The maximum number of retries for the SRQ can be limited, so that the data packet will only be dropped after this criteria is reached.

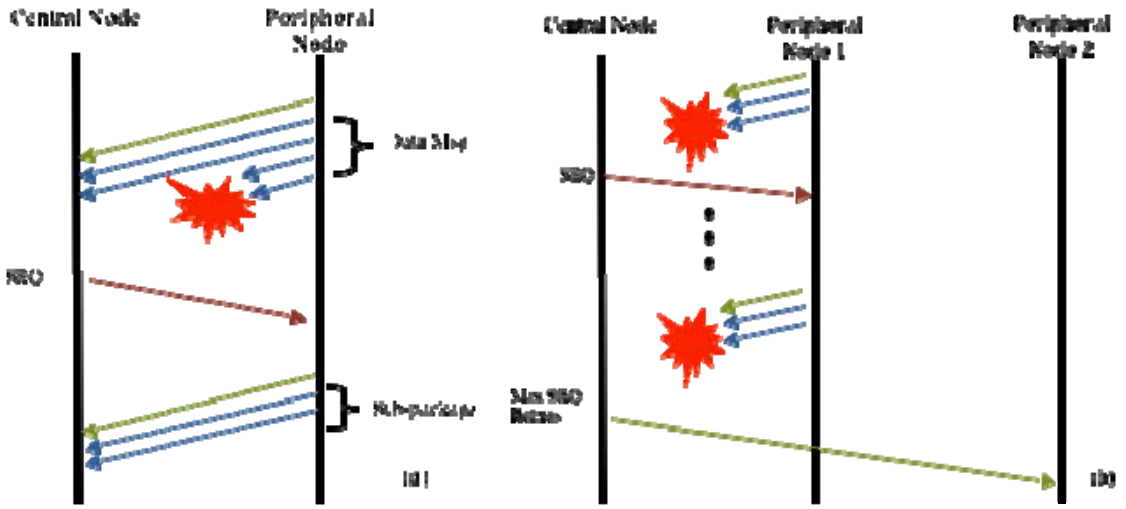


Figure 10. A data packet is divided into 16 sub-packets, each with its own CRC. When a corrupted data packet is received, the central node issues an SRQ utility packet requesting the peripheral node retransmit only the corrupted sub-packets. If necessary, the central node issues additional SRQs until either all sub-packets are successfully received or a pre-determined number of retries are reached. If a pre-determined number of retries is reached, the data packet is dropped and the central node proceeds to the next peripheral node.

## B. POLLING

The star topology includes a central node and a number of peripheral nodes. The central node polls each peripheral node sequentially, as shown in Figure 11, to control the flow of data traffic.

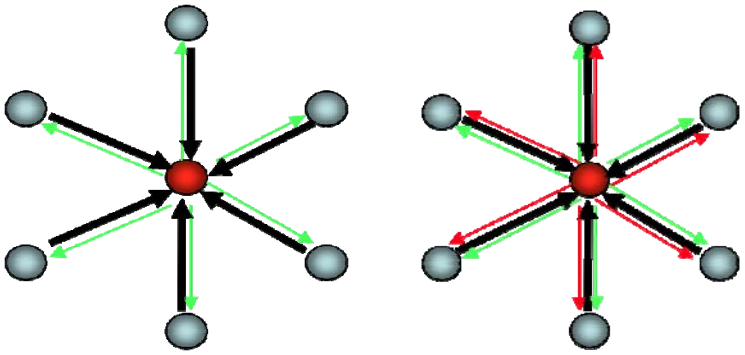

Figure 11. Star topology with polling protocol. The left diagram is without SRQ and the right is with SRQ. High-bit-rate data packets are depicted as wide arrows, and low-bit-rate utility packets as narrow arrows. (From [6])

### 1. Without SRQ

For the purpose of our simulation, when a corrupted utility or data sub-packet is received, the entire data packet is dropped. This policy ensures that the latency of the network remains low. The utility packet transmitted from the central node, due to its small size and low probability of error, is permitted for retransmission for a number of times. If the maximum allowed retries is reached, the data packet is dropped and the next node is polled.

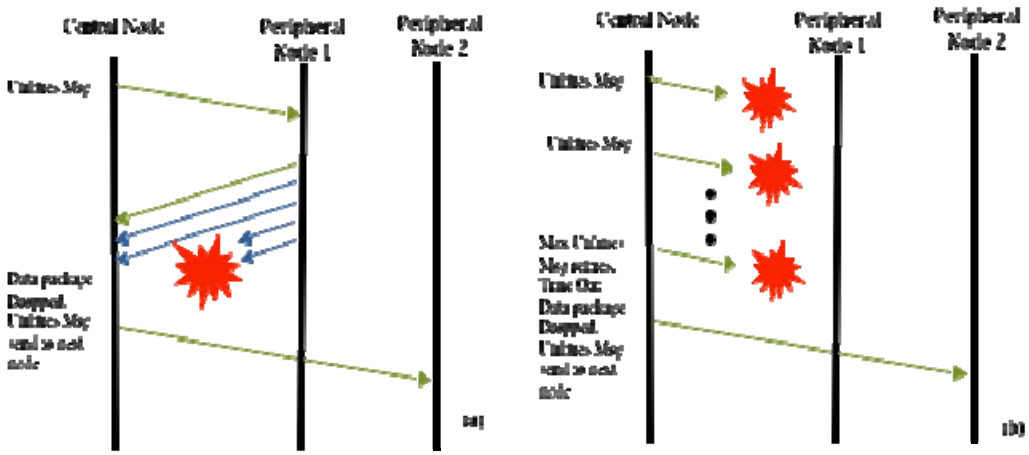

Figure 12. Polling without SRQ.

### 2. With SRQ

For polling with SRQ, as shown in Figure 13, the central node issues an SRQ when a corrupted utility packet or data sub-packet is received from the peripheral node. SRQ will continue to be sent to the same node until the maximum number of SRQ retries is reached or a complete data packet is received from that node. A new utility packet is then sent to the next peripheral node in line.

A silence from the peripheral node after issuing a utility packet by the central node can be due to either failure in the utility packet reaching the peripheral node or a failure in transmission from the peripheral node. The period of time from the transmission of the utility packet by the central node to issuing an SRQ can be adaptively calculated by computing the round-trip propagation delay ($2t_{PD}$), dwell time ($t_{Dwell}$) and the utility ($t_{Util}$) and data packet transmission ($t_{Data}$) times. They are defined as follows:

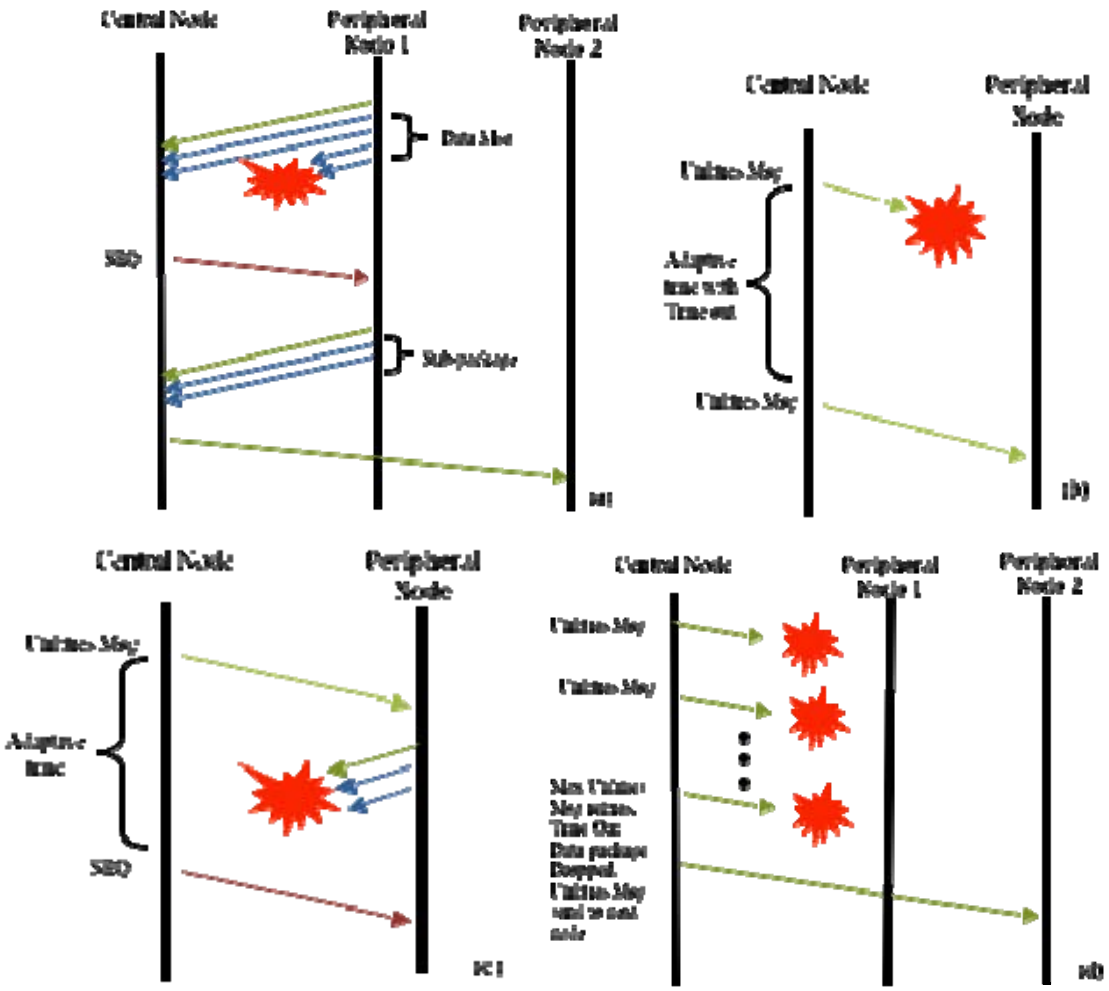


Figure 13. Polling with SRQ.

### *a. Propagation Delay*

Propagation delay is the time taken for the signal to travel from one node to the other. It is computed as Range * Speed of Sound. The range between the nodes is found during the network initialization phase by making use of a ranging function in the modem. The description of that function can be found in [14]. For our simulation, we assume the speed of sound over depth in seawater to be $c$ = 1500m/s [15].

### *b. Dwell Time*

The dwell time is the time from the arrival of the utility packet (poll) from the central node to the transmission of the utility packet (header) to the central

node. It consists of the modem wake-up time and computational time. It is set to a constant of 1.2 seconds in the simulation.

### *c. Utility Packet Transmission Time*

The time taken to transmit a utility packet is the Number of Bytes in the Utility packet * Data Rate of the Utility Packet. For the simulation, they are 9 bytes and 140 bits/second, respectively, for the simulation.

$$9 * 8 / 140 = 0.51 \text{ s} \tag{4.1}$$

### *d. Data Packet Transmission Time*

The time taken to transmit a data packet is the Total Number of Data Bytes * Data Rate of Data Packet. For our simulation, each data packet consists of 4000 data bytes and 96 overhead bytes, and they were transmitted at a data rate of 2400 bits/second.

### *e. Adaptive Time-Out Period*

The adaptive time-out period, as shown in Figures 12b and 12c, is calculated as:

$$T_{time-out} = 2 * t_{PD} + t_{Dwell} + t_{Util} + t_{Data} \tag{4.2a}$$

$$T_{time-out} = 2 * Range * c + t_{Dwell} + t_{Util} + t_{Data} \tag{4.2b}$$

where $c$ is the speed of sound in water.

## C. TOKEN RING

Unlike the polling protocol, the central node in the token ring network relinquishes control of the network flow by releasing a token to a peripheral node. The node that has the token performs its task (i.e., transmitting the data) before passing the token to the next node. This protocol should have a lower overhead compared with polling.

### 1. Without SRQ

In a token ring network that does not use SRQ, the central node releases the token to the peripheral nodes upon initialization and does not regain control of the token, as shown in Figure 14a. The central node continues to keep track of the position of the token. In the event that a token transmission between two nodes is lost, the central node intervenes by sending a new token to the next peripheral node in the line, as shown in Figure 14b.

Any corrupted data packet is dropped. This allows the network to perform with lower latency and shorter cycle time. The disadvantage of this protocol is that in a high noise environment, there is the potential of losing a high amount of data.

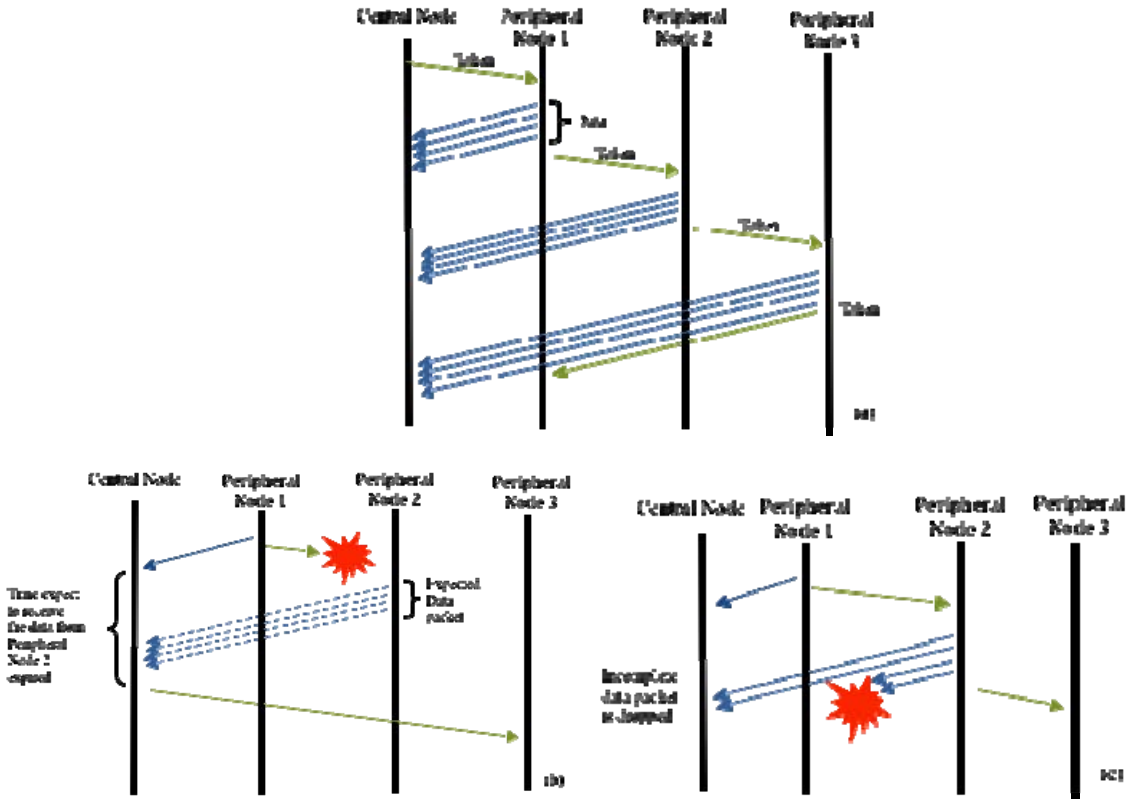


Figure 14. Token Ring Network without SRQ.

### 2. With SRQ

The final protocol is a combination of the token ring and polling with SRQ. In this case, as shown in Figure 15a, the central node takes note of failed data packets in addition to keeping track of the position of the token as before. When the token reaches the last of the peripheral nodes, it passes back to the central node which issues SRQs to attempt to reconstitute the corrupted data packet, node by node. A data packet is dropped after the maximum number of retries. After the central node completes the SRQ process, the token is again released to the peripheral nodes for a new cycle of data transfer.

This protocol helps in lowering the dropped data packet rate while keeping the latency level lower than with the polling with SRQ protocol.

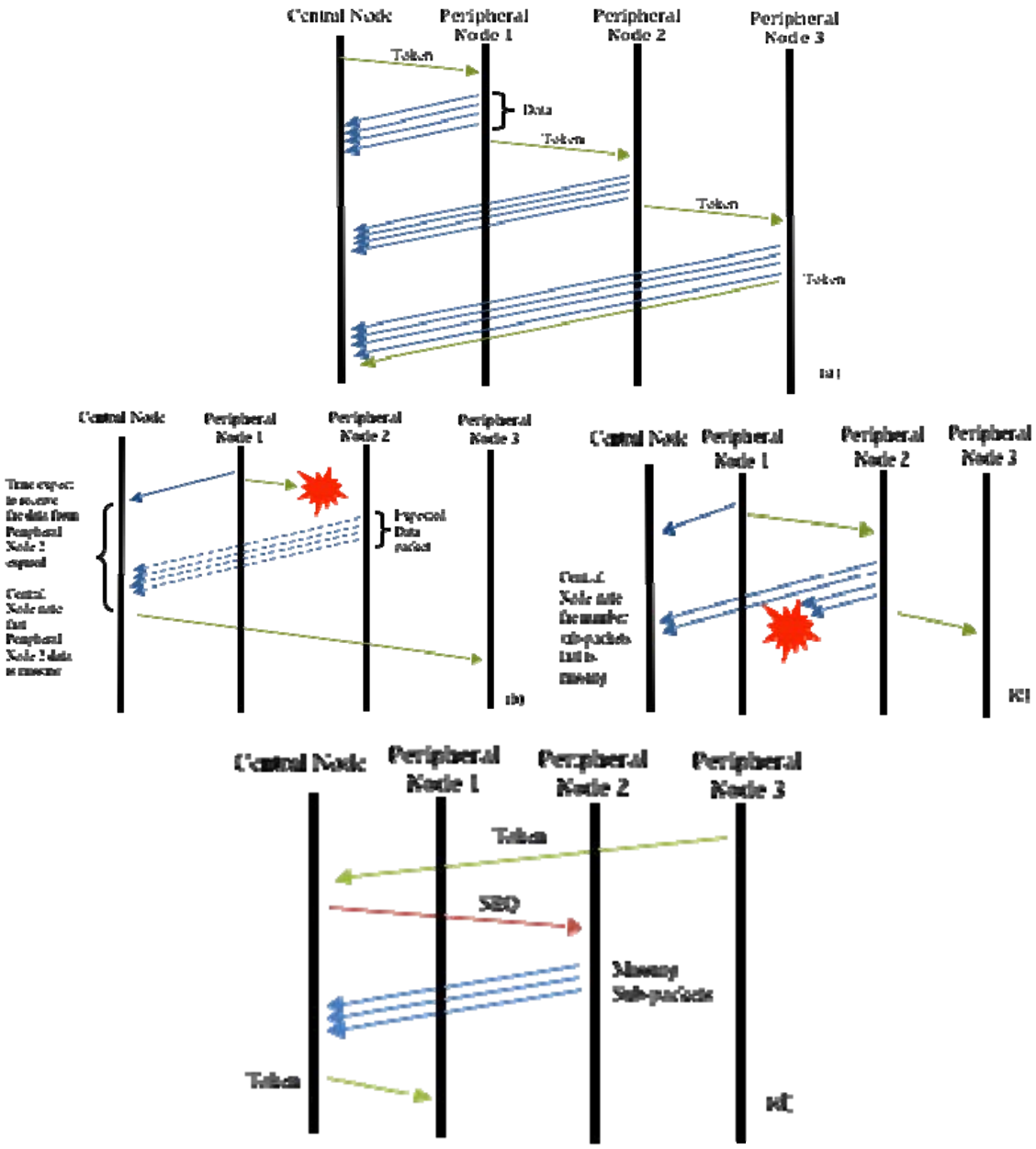


Figure 15. Token Ring network with SRQ.

THIS PAGE INTENTIONALLY LEFT BLANK

# V. NETWORK SIMULATION

Computer simulation of an event-driven Seastar network allows for controlled study of the four candidate network protocols. The different network protocols are exercised in the simulation through various parametric test cases. The simulations generate four measurements, namely the network latency, data throughput, dropped packets, and error-free cycle. For each test case, the candidate protocols are evaluated to determine which protocol is best suited for meeting particular operational requirements.

## A. SIMULATION PARAMETERS

### 1. General Definitions

Table 5 provides definitions of the terms used in this chapter and values assumed for the simulation.

| Type | Definition |
|---|---|
| Data Packet | Includes 16 sub-packets, containing a total of 4096 bytes, of which 4000 are data bytes and 96 overhead bytes. |
| Sub-Packet | Includes 250 data bytes and 6 overhead bytes. |
| Utility Packet | Contains 9 bytes. Used for regulating the flow of data traffic. Note that Utility packets will always precede the Data packet from the peripheral node to the central node. |

Table 5. Definition of terms used in the simulation.

The simulation is event-driven with a time increment between each event equivalent to the duration of the process occurring at that time. Table 6 provides the various events that are used for the simulation.

| Type | Duration | Description |
|---|---|---|
| $t_{propdelay}$ | Varies | Propagation delay between each pair of nodes is proportional to the range by the relation $t_{propdelay}$= r * c = r * 1500 m/s. |
| $t_{CPU}$ | 1.200 sec | Dwell time between receptions of utility packet to response. |
| $t_{TO}$ | 1 sec | Time out period. |
| $t_{Data}$ | 13.653 sec | Transmission time for one data packet containing 4096 bytes. |
| $t_{SubData}$ | 0.853 sec | Transmission time for one data sub-packet containing 256 bytes. |
| $t_{Util}$ | 0.450 sec | Transmission time for one utility packet. |

Table 6. Duration of various events.

## 2. Input Parameter

The simulation has two input parameters: position of the nodes in Cartesian coordinates, and the error rate of the data packet.

### *a. Node Position*

The node positions are specified within a 1001 by 1001 square grid with the origin [0,0] at the center of the grid. The central node is placed at the origin. The simulation calculates the range of the peripheral nodes to the central node and arranges them in order of their distance from the central node, from the nearest to the farthest. This is the sequence used by the polling protocol to access the peripheral nodes.

For the token ring protocol, the sequence of the nodes is further optimized by an algorithm that computes the shortest path to cycle through all the nodes, beginning and ending at the central node. The optimization algorithm limits the number of peripheral nodes to less than nine peripheral nodes for this simulation program.

Propagation times between nodes, $t_{propdelay}$, are calculated using the distances provided in Table 6.

### *b. Error Rate*

The input for data packet error rate ranges from 0.0 to 0.8; i.e., 0.4 at about $SNR_b$ = 10.4 dB, as shown in Figure 9. The simulation takes the data packet error

rate and computes the utility packet error rate at 400 times lower, as explained in the previous chapter; i.e., 0.001 at about $SNR_b$ =10.4 dB.

A random number between 0 and 1 of uniform distribution is generated at each event when a packet is transmitted. A corrupted utility or data packet is declared if the random number falls below the utility or data error rate, respectively. The simulation takes the appropriate measures when an error event occurs according to the policy prescribed by each network protocol.

## B. PERFORMANCE METRICS

Simulated operation of the Seastar network is assessed according to certain performance metrics related to important functional characteristics. Four such metrics are identified in the following sections.

### 1. Latency

Latency is a measurement of the time required to complete a cycle. Each cycle includes the transmission time, propagation delay, and dwell time. The latency measurement starts with a node issuing a utility packet (poll or token) to the next node and ends when the same node next issues the utility packet to the next node after completing all the nodes in the network. The latency measurement starts at the central node for the token ring with SRQ, as shown in Figure 16. The measurement starts at the first peripheral node for the token ring without SRQ.

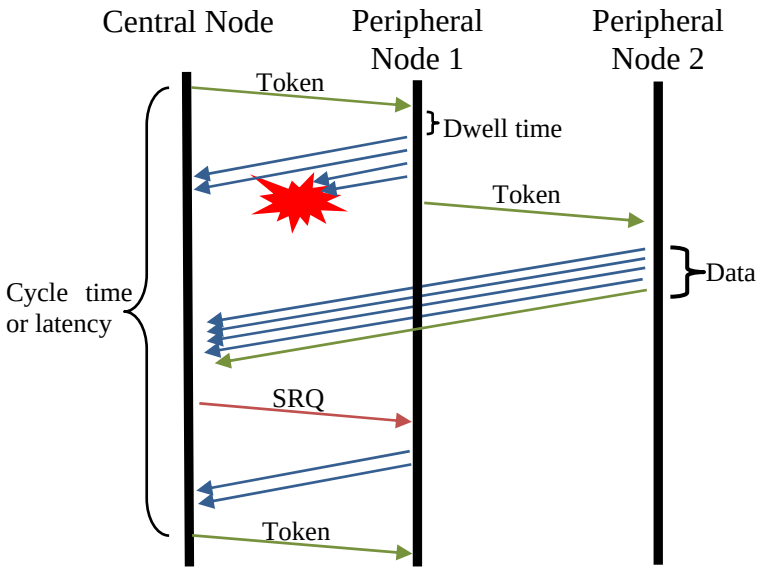


Figure 16. Calculation of latency for the token ring with SRQ protocol.

Two different measurements of latency are recorded, the average time per cycle, and the maximum time per cycle. The average time per cycle is computed by dividing the total simulation time by the total number of cycles completed. The maximum time per cycle is the longest time it takes to complete one cycle during a single simulation run. If there is more than one simulation run, an average of the maximum time is recorded.

### 2. Throughput

Throughput is a measure of the number of successfully transmitted data packets and data bytes the network can support within the simulation time. If SRQ is implemented, only the successful transmission of all sub-packets is counted as a successful transmission. If a dropped packet or timeout occurs, that packet is not considered a successful transmission.

### 3. Dropped Data Packet

A dropped data packet occurs in three different ways. In the two protocols without SRQ, in the event that an error bit occurs in the data packet, the data are considered corrupted and the packet is dropped. In the two protocols with SRQ, in the event that the maximum number of SRQ retries is reached, time-out is declared and the data packet is

dropped. For both polling protocols, in the event that the maximum number of retries of the poll utility packet is reached, the data packet for that peripheral node is dropped.

#### 4. Error-Free Cycle

Seastar is intended to gather data from each peripheral sensor node for assimilation at the central node upon completion of a cycle. Therefore, any occurrence of a dropped packet or timeout in a cycle may limit the successful fusion of data in that cycle. The error-free cycle throughput measures each complete error-free collection of data packets from all peripheral nodes in a cycle.

## C. CASE STUDIES AND PARAMETRIC ANALYSIS

There are three variables in the simulation. They are (1) number of nodes, (2) geometrical layout, and (3) the error rate. The variable error rate is used to simulate a low-, middle-, and high-noise environment. Simulation can show the sensitivity of the candidate protocol to these variable parameters.

### 1. Number of Peripheral Nodes

The simulation runs test the number of peripheral nodes = [2,3,4,5,6,7,8]. This parameter is useful in examining the impact of increasing the number of nodes on the different candidate protocols. The nodes are located on both a linear layout and a circular layout as described in the following section.

### 2. Layout of Nodes

#### *a. Linear Layout*

A single line layout is useful in various applications, such as a surveillance tripwire at a harbor entrance or a magnetic sensor network. Because the Seastar modem is designed to operate within a range of 500 m, the maximum aperture of a linear layout is 1 km if the central node is placed at the middle.

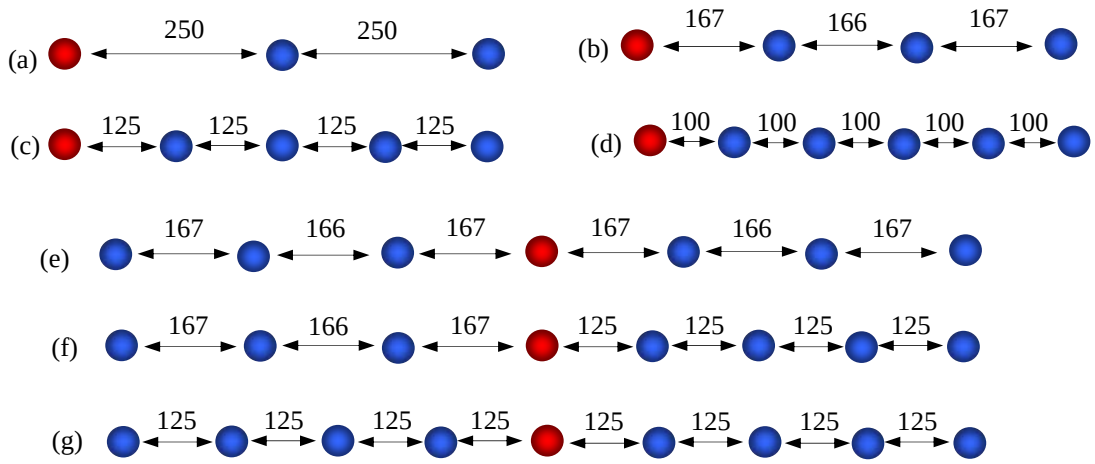


Figure 17. Different layouts of straight lines with two to eight peripheral nodes.

### *b. Circular Layout*

Circular layouts, as shown in Figure 18, serve as bounding cases for layouts involving a two-dimensional distribution of nodes. As described in [6], if there are fewer than six peripheral nodes in the network, the area coverage of a circular layout is limited by the assumed maximum communications range $r_{\max} = 500$ m. If there are six or more peripheral nodes, the coverage is limited by the central node when it is at the center of the circle. Therefore, the maximum communication coverage of the network when six or more peripheral nodes are deployed is $\pi r^2 = \pi(0.5)^2 = 0.785 \text{ km}^2$.

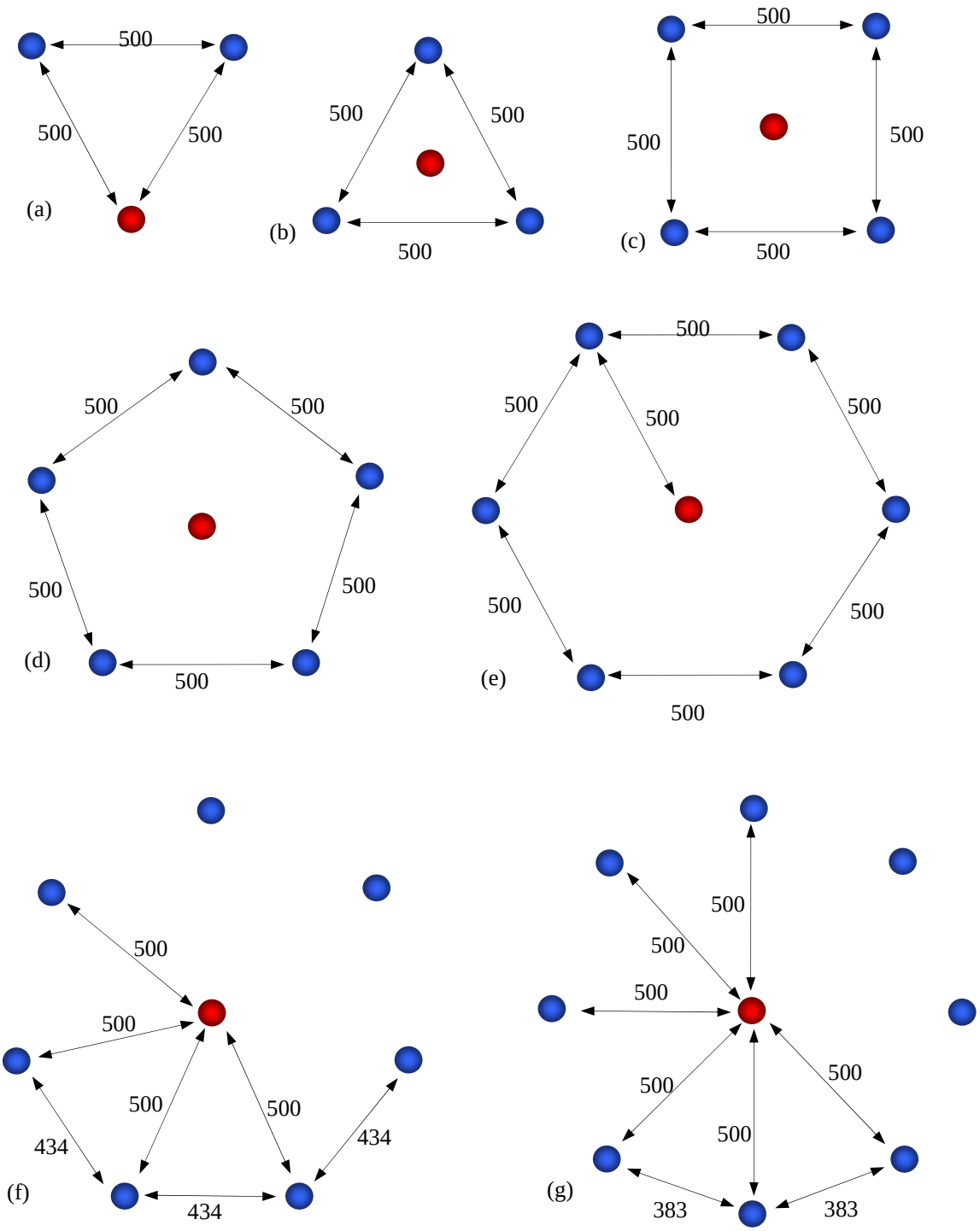


Figure 18. Different circular layouts with two to eight peripheral nodes. Note that the range between peripheral nodes begins to decrease when the total number of peripheral nodes exceeds six.

### 3. Error Rate

The error rate is used to investigate the effect of noisy environments on the network. This test parameter yields information on the relative performance of the protocols in good and bad communication environments.

Deployed networks experience a wide range of ambient noise conditions. In chapter III, it was estimated that the $SNR_{Data\_Min}$, $SNR_{Data\_Max}$ $SNR_{Util\_Min}$ and $SNR_{Util\_Max}$ for the data and utility packet to be -8.4 dB, 23.1 dB, 3.4 dB and 34.9 dB, respectively. This gives an *SNR* range of 31.5 dB.

In order to achieve a data packet error rate of less than $10^{-5}$, $SNR_b > 13$ dB is required, as shown in Figure 9. When $SNR_b < 10.3$ dB, a data packet error rate of more than 0.5 is generated. For the purpose of this simulation, these two values will be considered the low- and high-noise environments; that is, a Noise Level of less than 73 dB is considered to be a low-noise environment, and a Noise Level of greater than 75.8 dB is considered a high-noise environment. A data packet error rate of 0.2 is used to simulate a mid-noise environment.

# VI. RESULTS

## A. EFFECTS OF NUMBER OF NODES AND GEOMETRICAL LAYOUT

Figure 19 displays the result of a 0.0 error rate with the number of peripheral nodes, N = [2,3,4,5,6,7,8] in both linear and circular layouts. The average time per cycle plot shows that both polling methods have the same performance, since there are no bit errors occurring in the simulation. They exhibit the longest average cycle time due to the overhead of polling. The token ring with SRQ is the next longest, as the token unnecessary passes through the central node. The token ring without SRQ provides the shortest cycle time. The cycle times increase linearly as the number of nodes increase, which is to be expected.

For the maximum cycle time plot, it might be expected to see the same plot as the average time plot, as there are no errors to prolong the cycle time. However, the two token ring protocols lay on the same line. This is because for the token ring without SRQ, the token begins from the central node, and this first cycle time is the longest, and is thus registered.

In the throughput plot, we observe the geometrical layout of the nodes coming into play. In general, the linear layout enables the network to carry more data packets, as the distances between the central and peripheral nodes, as well as between the peripheral nodes themselves, are shorter when compared to the circular layout. The only exception to this is the 3-peripheral circular layout when polling with SRQ protocol is applied. This is because the central node is in the center of the triangle, as shown in Figure 18b, and therefore, has a shorter traveling distance to the peripheral nodes. This enables it to have a higher packet rate.

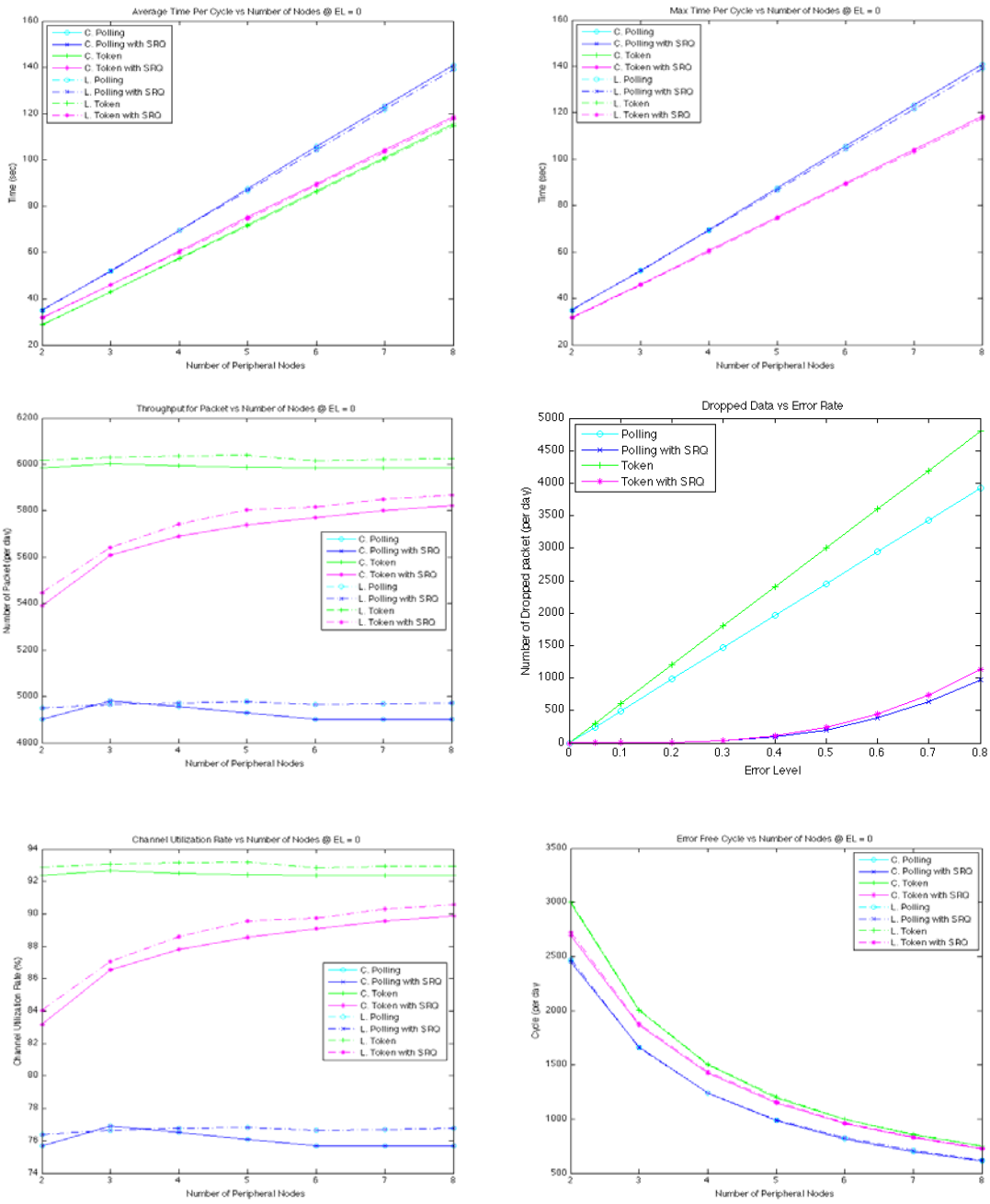


Figure 19. Number of peripheral nodes with an error rate of 0.0.

It can be observed that the token without SRQ protocol provides the most throughput in an error-free environment, as it has the least overhead. It is followed closely by token with SRQ. Additionally, both of the polling protocols are the least efficient. The same trend is seen in the channel utilization rate.

Finally, in the error-free cycle plots, as the number of peripheral nodes exceeds six, the coverage of the layout does not change, due to the star topology. Only the density of the nodes in a given area increases with more than six peripheral nodes. Therefore, the number of error-free cycles decays exponentially, as seen in the plot.

Figure 20 displays the performance with a change in the error rate parameter from 0.0 to 0.2 applied over the same number of nodes, as in Figure 19.

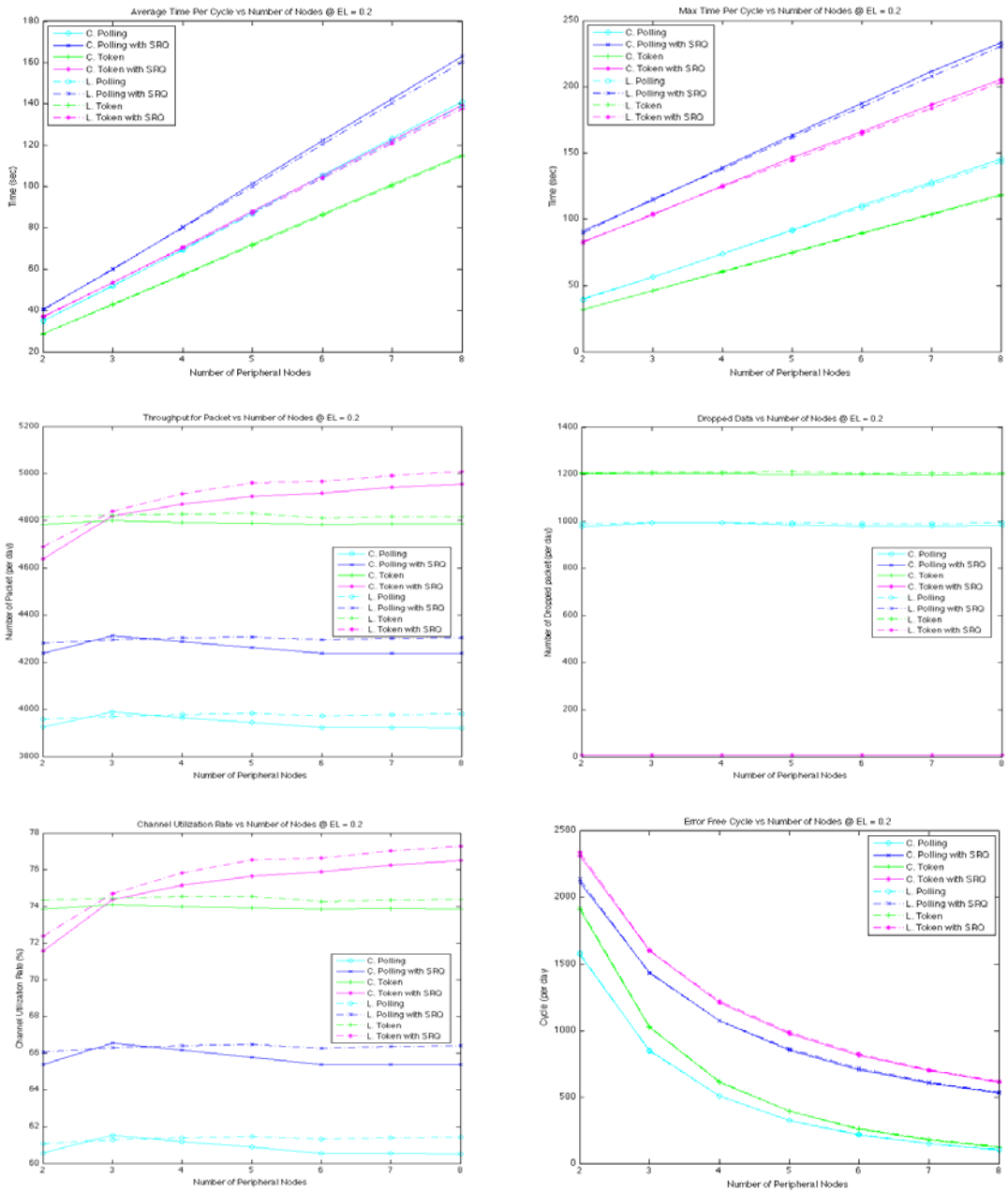


Figure 20. Number of peripheral nodes with error rate of 0.2.

When errors are introduced into the network, some important changes occur. The latency of the protocols with SRQ increases due to the additional error correction mechanisms at work. For those protocols that do not implement SRQ, dropped packet

rate increases significantly, as no error correction efforts are made with no change in latency. All the protocols experience a reduction in the error-free cycle. Those protocols that do not implement SRQ are the most adversely affected.

In concluding this section, we observe that as the latency of the network is both a function of the number of nodes as well as the density of the nodes in a given area. Thus the closer the peripheral nodes are to the central node, the lower the latency. Circular layouts, being the bounding geometry, show the poorer performance overall as the distance between the nodes are greater. In an error-free network, protocols without SRQ allow the network to operate at maximum efficiency, but when errors are introduced into the network, protocols with SRQ enable the network to continue to function with less degradation.

## B. EFFECT OF ERROR RATE

This section shows the effect of varying the error rate from 0.0 to 0.8 in a six-peripheral network. This variation of error rate is to simulate changing environmental noise and its impact on the network. As noted in the previous section, the circular layout is the bounding case, and this section will only address the circular layout.

### 1. Time Per Cycle Versus Error Rate

In Figure 21, it is observed that for the protocols without SRQ, both the average time per cycle and the maximum time per cycle remain constant with increasing error rate. This is because whenever there is a corrupted data packet, it is dropped with no delay to the network. For the protocol with SRQ, latency increases as the error rate increases because of the time consumed by the SRQ mechanisms.

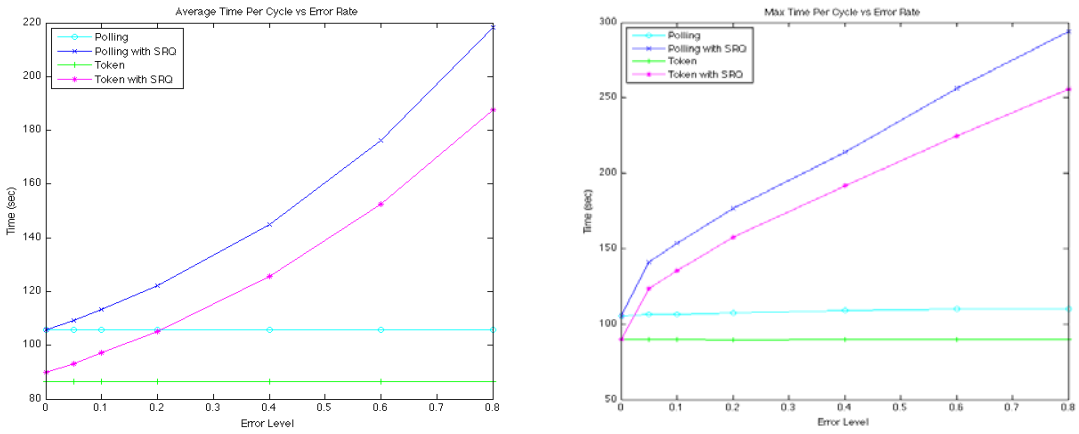


Figure 21. Average and maximum time per cycle verse error rate.

## 2. Throughput and Dropped Data Rate

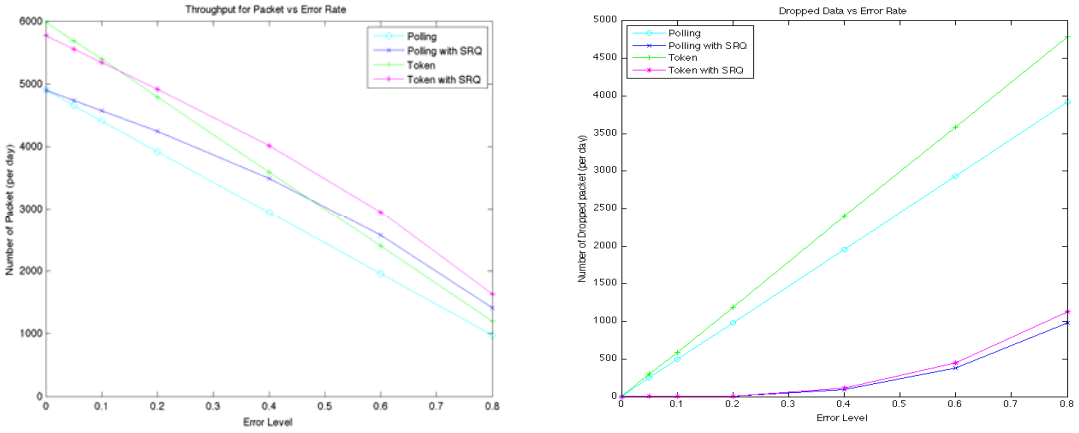


Figure 22. Throughput and dropped data rate versus error rate.

In the throughput plot in Figure 22, it is evident that the plot representing token ring without SRQ crosses the token ring with SRQ and polling with SRQ at error rates of about 0.12 and 0.45, respectively. Therefore, if maintaining a high level of throughput is a priority and dropped data is of less concern, network protocols should be switched when an increase in ambient noise level is observed.

In the dropped data plot, the beginning of an exponential increase in dropped data at error levels above 0.2 becomes evident. This rate can be suppressed by increasing the maximum number of SRQs. However, this adversely impacts the latency of the network, and is therefore a trade-off.

### 3. Channel Utilization and Error-Free Cycle

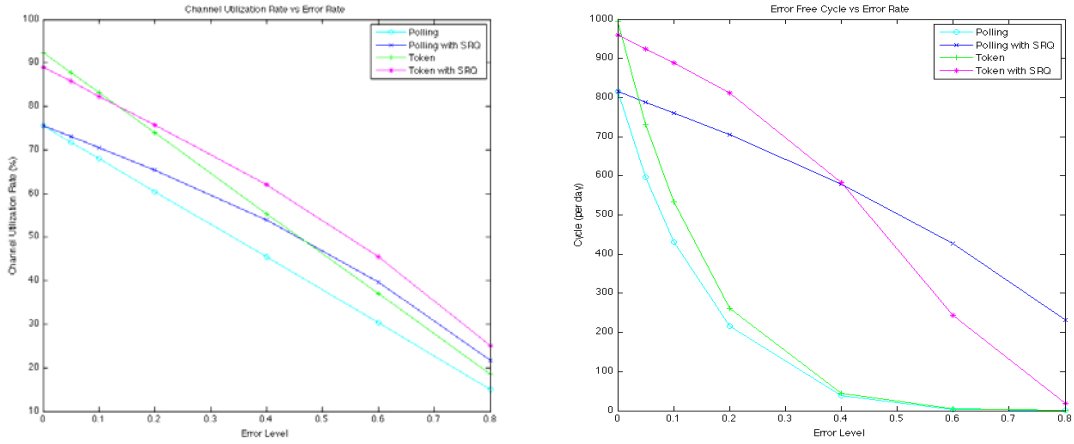


Figure 23. Channel utilization and error-free cycle versus error rate.

The channel utilization plot in Figure 23 illustrates how the utilization rate decreases with increasing error rate as more and more of the channel availability is used by error correction. In many applications of Seastar, an error-free cycle is important if the central node requires uncorrupted data from all peripheral nodes to perform data fusion for the cycle. From the error-free cycle plot in Figure 23, a crossover point between token with SRQ and polling with SRQ at an error rate of 0.4 occurs. This indicates that the polling with SRQ protocol is more robust at maintaining error-free cycles in noisy environments.

THIS PAGE INTENTIONALLY LEFT BLANK

# VII. CONCLUSION AND RECOMMENDATIONS

## A. SUMMARY OF FINDINGS

Seastar is a LAN design for accumulating underwater distributed sensor data at a central node for data fusion. This thesis evaluated four candidate network protocols using an event-driven simulation. Figure 24 ranks the candidate protocol according to their relative performance against particular metrics.

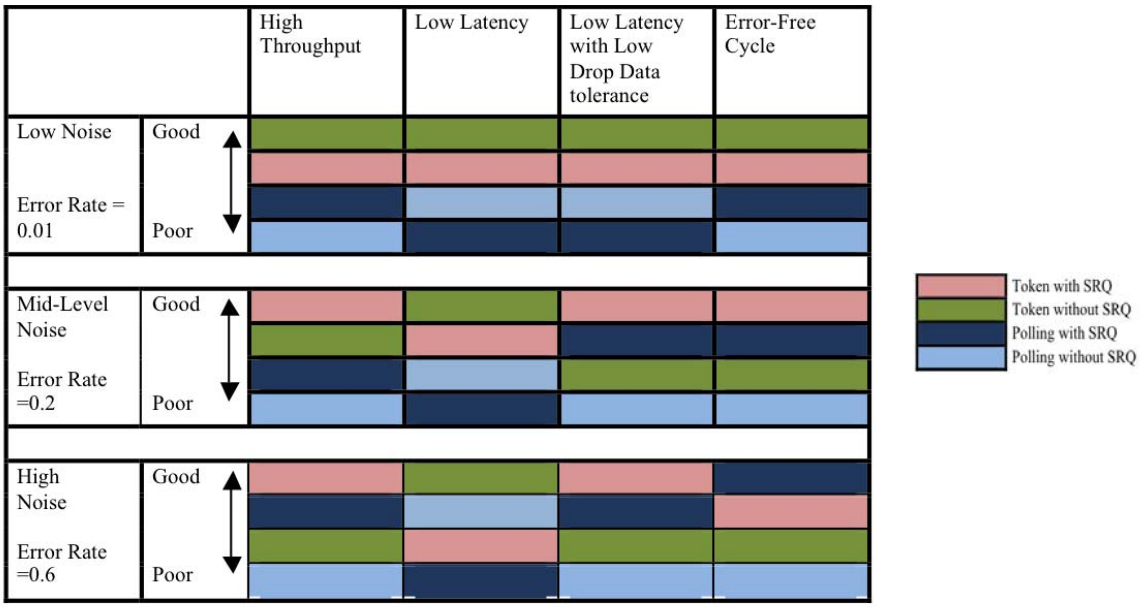


Figure 24. Performance metrics

For high throughput performance in low noise conditions, the token ring without SRQ performs the best while in mid to high noise conditions, the token ring with SRQ is superior. For low latency performance where dropped data packets can be tolerated, token ring without SRQ provides the lowest cycle time. However, if there is a need to minimize the number of dropped data packets, token ring with SRQ is superior.

In a data-sensitive network, where error-free cycles are necessary for successful data fusion, the token ring with SRQ performs well from low to mid of noise. However, in high noise environments, polling with SRQ performs better than token ring with SRQ.

Polling without SRQ does not have any advantage over the other three protocols in any scenario considered here. Therefore, it is recommended that this protocol not be considered for future implementation.

For the simulation, all the modems were considered to be in either listening or transmitting mode without entering a low-power mode. This means that battery energy is constantly being consumed. In a scenario where the sensor data refresh rate is much longer than the cycle time, having the modem switch to low-power mode could conserve energy. Under this condition, the traditional protocol used by Seaweb, polling with SRQ, would provide both a centrally controlled data transfer in conjunction with fast and reliable throughput.

## B. CONCLUSION

Seastar is a viable networking concept. The network protocols considered here are well suited for the high throughput and reliability required by applications such as maritime surveillance. The centralized topology is advantageous in that only the central node need be sophisticated enough to perform data fusion, receive and process high bit-rate data packets, orchestrate the LAN operations, and communicate through the Seaweb WAN or gateway node. From a communication standpoint, the peripheral nodes can be relatively simple, capable only of receiving utility packets. Such limited capability at the peripheral nodes permits them to receive commands from the central nodes (e.g., poll and SRQ) and peer-to-peer communications from neighboring peripheral nodes (e.g., token and ping/echo).

It is concluded from this thesis research that the networking protocol should include multiple operating nodes to be invoked under the control of the central node. The selected protocol should be matched to the operational requirement (e.g., throughput, latency, reliability) and the environmental conditions (e.g., noise conditions and error rates).

## C. RECOMMENDATIONS FOR FUTURE WORK

### 1. Energy Conservation

The acoustic modem obtains its power from battery cells and this usually limits the operational time. This thesis estimates the maximum throughput and minimum latency by keeping the modem constantly in an "awake" mode. Therefore, this study does not allow energy conservation to play a part in protocol selection.

In the event when the peripheral node refresh rate is low, the modem could be placed in a low-power state between each cycle, thereby conserving its energy. There is a need to evaluate the potential energy savings against the requirement for latency, throughput and reliability.

### 2. Environmental Noise Simulation

The simulation conducted in conjunction with this thesis assumes static levels of error rate. To develop this simulation program  a mission planning tool, a module that simulates or accepts a playback of pre-recorded noise level fluctuation characteristic of the intended operational area, should be added. This would provide the mission planner and operator with a better understanding of how the network will perform.

### 3. Performance Measurement of Seastar Modems

The transmitted power and source level for this simulation is obtained from the vendor specification sheet. As of June 2010, the Seastar modems are undergoing testing and measurement by another student at the Naval Postgraduate School (NPS). Once the testing is complete, the actual modem data could be used to refine the simulation and its test products.

THIS PAGE INTENTIONALLY LEFT BLANK

## APPENDIX SIMULATION ALGORITHM IN MATLAB

This function iniSeaStar.m is called in the seastarsim.m function and is to initialize the Seaweb simulation with the required data parameter.

```
function [SSPack] = iniSeaStar()


N = 7; %Number of Modems including the central node
Data_Mod = 2400; %Data modulation in bits per sec
Util_Mod = 160; %Utility modulation in bits per sec
Sound_Speed = 1500; %In meters per sec
Poll_CTS = 9; %Number of bytes used for Polling
Data = 4000; %Number of bytes
Processor_Delay = 1200; %Time for each modem to process the data in msec


%Parameter settings
Token_Byte = 9; % 2 bytes for addressing, 2byte for CRC, 1 byte for
%each node and 3 byte for SRQ addressing for each node.
Data_Mod = Data_Mod/8; % Data modulation in bytes per sec
Util_Mod = Util_Mod/8; % Utilities modulation in bytes per sec
SubPack = ceil(Data/16)+6 ; % 6 information bytes per subpacket
DataPack = SubPack*16; %16 Subpackets make up 1 data packet
Time_DByte = 1/Data_Mod; %Time to send a byte of data
Time_UByte = 1/Util_Mod;
Time_UtsPack = Time_UByte*Poll_CTS;      %Time for Tx CTS in sec
Time_TokenPack = Time_UByte*Token_Byte;
Time_DataPack = Time_DByte*DataPack;     %Time for Tx data in sec
Time_DataSubPack = Time_DByte*SubPack;
Time_Pro = Processor_Delay/1000; %Time Delay of Processor in sec

SSPack = [Time_DataPack,Time_DataSubPack,Time_UtsPack,Time_Pro,...
   Sound_Speed,N,DataPack,Poll_CTS,Time_TokenPack,Token_Byte,SubPack,...
   Data,Data_Mod,Util_Mod];
```

//////////////////////////////////////////////////////////

This function GridNode.m is called in the seastarsim.m function and takes in two variable:

Number_of_Nodes = total deployed nodes (N)

Matrix_of_Grid = A two by Number of nodes matrix with row 1 as the x-axis and row 2 as the y-axis of the grid. The first column is the central node position.

The function will generate a grid of 1001 by 1001 and position the node into the grid. The central node will be place at position (0,0), while the rest will take on a position from -500 to 500 according.

The function will return an N+2 by N matrix with Rows 1 and 2 containing the x and y coordinates of the position grid of the nodes, respectively. Columns 1 to N are the node rankings by distance away from the central node with Column 1 being the central node position. Rows 3 to Row N+2 versus Column 1 to N is the distance between those nodes.

```
function [Block] = GridNode(Number_of_Nodes,Matrix_of_Grid)


N = Number_of_Nodes;
M = Matrix_of_Grid;
B = zeros(N+2,N);

for i = 1:2
   for j = 1:N
      if i == 1
         B(i,j) = M(i,1)-M(i,j);
      else
         B(i,j) = M(i,1)-M(i,j);
      end
   end
end

%Distance calculation.

for i = 3:(N+2)
   for j = 1:N
      B(i,j)=sqrt((B(1,i-2)-B(1,j))^2+(B(2,i-2)-B(2,j))^2);
   end
end

% Sort row 3 and column 1 of matrix according to distance from central node.
B = sortrows(B,1);
Block = rot90(sortrows(rot90(B,-1),N));
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function PropTime.m calculates the propagation delay between the nodes. Input includes the number of nodes, sound speed and the grid layout of the network. Output of the function <Proptime> will consist of the grid position of the node and the propagation time delay between the nodes.

```
function [TPropT] = PropTime(N,SS,Grid)

SpeedMatrix = cat(1,ones(2,N),repmat(SS,N,N));

TPropT = Grid./SpeedMatrix;

maxSpeed = 502/SS;

for i= 3:(N+2)
   for j = 1:N
      if TPropT(i,j) > maxSpeed
         TPropT(i,j) = 999;
      end
   end
end
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function TkRing.m is called in the seastarsim.m function. It is used to order the nodes in for the Token Ring network that will provide the lowest amount of delay. It take in the input of the Matrix of TPropDelay and Number of Nodes and outputs the Token Order <Torder> and Token Ring Propagation Delay <TRdelay>

```
function [Torder,TDorder] = TkRing(TPD,Node)

a = sortrows(perms(2:Node),1:Node-1);
[x,y] = size(a);
b = ones(x,y+2);
b(:,2:y+1)=a;

TD = TPD(3:Node+2,:);

TDwithOrder = zeros(x,Node);

for i = 1:x
```

```
    for j = 1:Node
        loc = b(i,j:j+1);
        TDwithOrder(i,j) = TD(loc(1),loc(2));
    end
end

TotDelay = sum(TDwithOrder,2);
[TRdelay,Indexline] = min(TotDelay);

Torder = b(Indexline,:);
TDorder = zeros(2,Node);
TDorder(1,:) = TDwithOrder(Indexline,:);
for i = 2:Node
    b = Torder(i);
    TDorder(2,i) = TD(1,b);
end
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

The function Polling2.m is called in the seastarsim.m function and is used to simulate the polling network with SRQ.

```
% Input of <Polling2> consists of the following
% a. ST = Simulation Time
% b. SSpack(1) = TDP = Tx Time for a Data Packet
% c. SSpack(2) = TDSP = Tx Time for a Data Subpacket
% d. SSpack(3) = TUP = Tx Time for a Utilities packet
% e. SSpack(4) = TCPUD = Delay time for modem CPU
% f. SSpack(6) = N = Number of nodes
% g. SSpack(7) = DB = Number of Data bytes
% h. PPD = A array of Propagation delay between nodes in sequence 1 to N
% i. SSpack(8) = Number of polling utilities bytes
% j. SSpack (9) = TTP = Tx time for a token ring
% k. SSpack(10) = TRB = Number of Token Ring bytes
% l. SSpack(11) = SPB = Number of subpacket bytes
%
% Output of <Polling> will consist of the following:
% a. PSDP = Successful number of data packets
% b. PSUP = Successful number of Utility packets
% c. PSB = Successful number of byte Tx.
% d. PSPRT = Number of sub-packets re-transmitted.
% e. PFSP = Number of sub-packet failures with time of failure
% f. PFP = Number of packet failure with time of failure
% g. PFU = Number of failed utilities Tx failures with time log of failure
% h. PTO = Number of Time-outs
```

```
function [PSDP,PSUP,PSB,PSPRT,PFP,PFSP,PFU,PTO,CT] =
Polling2(ST,SSpack,PPD,EL)


%Ini Setup for function
time = 0;   % Run time
sctime = 0; % Start Cycle time
nCycle = 0; % Number of cycles
sCycle = 0; % Number of Successful Cycles
maxCycletime = 0; % max time per cycle
avgCycletime = 0; % Average time per cycle
NEvent = 0; % Running number of Events
PSDP = 0;   % Number of successful Data packets transmitted
PSUP = 0;   % Number of successful Utilities packets transmitted
PSB = 0 ;   % Number of Bytes transmitted
PSPRT = 0;  % Number of sub-packets retransmitted
PFP = 0;    % Number of failed packets event
PFSP =0;    % Number of failed sub packet
PFU = 0;    % Number of failed Utility packet
PTO = 0;    % Number of Time-Outs
maxRetry = 3; % Number of retries before Time-out.
maxSRQ = 16; % Number of sub-packets
N = SSpack(6); %Number of Nodes
PDCount = 2; % Propagation Delay Counter
ELU = EL(1); % Error Level of Utilities
ELP = EL(2); % Error Level of Data Packet
TUP = SSpack(3);
TCPU = SSpack(4);
TO = 1; % TO wait
SPB = SSpack(11);
state = 1;
URC = 0; % Number of Utilities Retries
SRQRC = 0; % Number of SRQ Retries
MaxSRQR = maxRetry;
MaxUR = maxRetry;
SRQFlag = 0;
SRQerror = 0;
DropPackFlag = 0; % flagged if there's a dropped packet

%Main Program

while time <= ST
   switch state

      case 1
```

```
        [time,B,NEvent] = PCnode2(time,ST,SSpack,PPD(PDCount),NEvent);
        state = 2;

    case 2
        PSUP = PSUP + 1;
        PSB = PSB + B;
        if AnyError(ELU) == true
            state = 4;
            PSB = PSB - B;
            PFU = PFU + 1;
            PSUP = PSUP - 1;
            time = time + TCPU+PPD(PDCount)+TUP+TO;
        else
            state = 3;
            [time,B,P,SP,U,NEvent] =
PPnode2(time,ST,SSpack,PPD(PDCount),SRQerror,SRQFlag,NEvent);
        end

    case 3
        PSB = PSB + B;
        PSDP = PSDP + P;
        PSUP = PSUP + U;
        if SRQFlag == 1
            PSPRT = PSPRT + SP;
        end
        if AnyError(ELU) == true
              SRQerror = maxSRQ;
              if SRQFlag == 1
                 PFSP = PFSP + SRQerror;
              end
              PFU = PFU+1;
              PSB = PSB - B;
              PSUP = PSUP - U;
              state = 5;
         elseif AnyError(ELP) == true
            if SRQFlag == 1
              SRQerror = ceil(random('unif',1,SRQerror));
              state = 6;
              PFSP = PFSP + SRQerror;
              PSB = PSB - (SRQerror*SPB);
            else
              SRQerror = ceil(random('unif',1,maxSRQ));
              state = 6;
              PFSP = PFSP - maxSRQ + SRQerror;
              PSB = PSB - (SRQerror*SPB);
```

```
        end
        PFP = PFP +1;
        PFSP = PFSP + maxSRQ;
        SRQFlag = 1;
        PSDP = PSDP - P;
    else
        state = 7;
    end

case 4
    if URC == MaxUR
        state = 7;
        PTO = PTO + 1;
        DropPackFlag = 1;
    else
        URC = URC + 1;
        state = 1;
    end

case 5
    if SRQRC == MaxSRQR
        PTO = PTO + 1;
        DropPackFlag = 1;
        state = 7;
    else
        SRQRC = SRQRC+1;
        state = 4;
        time =time+TO; %util fail to reach back Central node, TO is the waiting gap
    end

case 6
    if SRQRC == MaxSRQR
        PTO = PTO + 1;
        DropPackFlag = 1;
        state = 7;
    else
        state = 1;
        SRQRC = SRQRC+1;
    end

case 7
    if PDCount == N
        PDCount = 2;
        nCycle = nCycle + 1;
        ctime = time - sctime;
```

```
            sctime = time; % start of new cycle
            if ctime > maxCycletime
               maxCycletime = ctime;
            end
            if DropPackFlag ~= 1
               sCycle = sCycle+1;
            end
            avgCycletime = (time/nCycle);
         else
            PDCount = PDCount+1;
         end
         URC = 0;
         SRQRC = 0;
         SRQFlag = 0;
         SRQerror = 0;
         DropPackFlag = 0;
         state = 2;
         [time,B,NEvent] = PCnode2(time,ST,SSpack,PPD(PDCount),NEvent);
   end
end

CT = [avgCycletime,maxCycletime,nCycle,sCycle];
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

The function of PollingS.m is called in the seastarsim.m function and is to simulate the polling protocol without SRQ

```
% Function for <PollingS> uses the following input
% a. ST = Simulation Time
% b. SSpack(1) = TDP = Tx Time for a Data Packet
% c. SSpack(2) = TDSP = Tx Time for a Data Sub-packet
% d. SSpack(3) = TUP = Tx Time for a Utilities packet
% e. SSpack(4) = TCPUD = Delay time for modem CPU
% f. SSpack(6) = N = Number of nodes
% g. SSpack(7) = DB = Number of Data bytes
% h. PPD = An array of Propagation delay between nodes in sequence 1 to N
% i. SSpack(8) = Number of polling utilities bytes
% j. SSpack (9) = TTP = Tx time for a token ring
% k. SSpack(10) = TRB = Number of Token Ring bytes
% l. SSpack(11) = SPB = Number of subpacket bytes
%
% Output of <Polling> will consist of the following
% a. PSDP = Successful number of data packet
% b. PSUP = Successful number of Utilities sign
```

```
% c. PSB = Successful number of byte Tx.
% d. PSPRT = Number of sub-packet re-transmitted.
% e. PFSP = Number of sub-packet failures with time of failure
% f. PFP = Number of packet failure with time of failure
% g. PFU = Number of failed utilites Tx failure with time log of failure
%
function [PSDP,PSUP,PSB,PFP,PFU,PTO,CT] = PollingS(ST,SSpack,PPD,EL)

%Ini Setup for function
time = 0;   % Run time
sctime = 0; % Start Cycle time
nCycle = 0; % Number of cycle
sCycle = 0; % Number of Successful Cycle
maxCycletime = 0; % max time per cycle
avgCycletime = 0; % Average time per cycle
NEvent = 0; % Running number of Events
PSDP = 0;   % Number of successful Data packet transmitted
PSUP = 0;   % Number of successful Utilities packet transmitted
PSB = 0 ;   % Number of Byte transmitted
PFP = 0;    % Number of failed packet event
PFU = 0;    % Number of failed Utility packet
PTO = 0;    % Number of Time-Outs
maxRetry = 1; % Number of retries before Time-out.
N = SSpack(6); %Number of Nodes
PDCount = 2; % Propagation Delay Counter
ELU = EL(1); % Error Level of Utilities
ELP = EL(2); % Error Level of Data Packet
TUP = SSpack(3);
TCPU = SSpack(4);
TO = 1; % Additional delay
state = 1;
URC = 0; % Number of Utility Retries
MaxUR = maxRetry;
DropPackFlag = 0; % flagged if there's a dropped packet


%Main Program

while time <= ST
   switch state

      case 1
         [time,B,NEvent] = PCnode2(time,ST,SSpack,PPD(PDCount),NEvent);
         state = 2;
```

```
case 2
   PSUP = PSUP + 1;
   PSB = PSB + B;
   if AnyError(ELU) == true
      state = 4;
      PSB = PSB - B;
      PFU = PFU + 1;
      PSUP = PSUP - 1;
      time = time + TCPU+PPD(PDCount)+TUP+TO;
   else
      state = 3;
      [time,B,P,SP,U,NEvent] = PPnodeS(time,ST,SSpack,PPD(PDCount),NEvent);
   end

case 3
   if AnyError(ELP) == true
      state = 5;
      PFP = PFP+1;
      DropPackFlag = 1;
   else
      PSB = PSB + B;
      PSDP = PSDP + P;
      PSUP = PSUP + U;
      state =5;
   end

case 4
   if URC == MaxUR
      state = 5;
      PTO = PTO + 1;
      PFP = PFP+1;
      DropPackFlag = 1;
   else
      URC = URC + 1;
      PFU= PFU+1;
      state = 1;
   end


case 5
   if PDCount == N
      PDCount = 2;
      nCycle = nCycle + 1;
      ctime = time - sctime;
      sctime = time; % start of new cycle
```

```
            if ctime > maxCycletime
               maxCycletime = ctime;
            end
            if DropPackFlag ~= 1
               sCycle = sCycle+1;
            end
            avgCycletime = (time/nCycle);
            DropPackFlag = 0;
         else
            PDCount = PDCount+1;
         end
         URC = 0;
         state = 2;
         [time,B,NEvent] = PCnode2(time,ST,SSpack,PPD(PDCount),NEvent);
   end
end

CT = [avgCycletime,maxCycletime,nCycle,sCycle];
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function PPnode2 is used by Polling2.m function to clock the simulation run time and the communication traffic from the peripheral nodes to the central node.

```
function [Clock,B,P,SP,U,NE] = PPnode2(Clock,ST,Pack,PPD,SRQ,SRQFlag,NE)

B=0;
P=0;
SP = 0;
U = 0;
TCPUD = Pack(4);
TDP = Pack(1);
TDSP = Pack(2);
TUP = Pack(3);
DB = Pack(7);
UB = Pack(8);
SPB = Pack(11);

Clock = Clock +TCPUD;
if Clock < ST
   Clock_S = Clock+TUP;
   if Clock_S < ST
      B = UB;
      U = 1;
```

```
      if SRQFlag == 0
         Clock_S = Clock_S+TDP;
         if Clock_S < ST
            B = B+DB;
            P = 1;
         else
            B = B+DB-ceil((Clock_S-ST)/(TDP/DB));
         end
      else
         Clock_S = Clock_S+(TDSP*SRQ);
         if Clock_S < ST
            B = B+(SRQ*SPB);
            SP = SRQ;
            P = 1;
         else
            B = B+(SRQ*SPB) - ceil((Clock_S-ST)/((TDSP)/(SPB)));
            SP = SRQ - ceil((Clock_S-ST)/TDSP);
         end
      end
   else
      B = UB - ceil((Clock_S-ST)/(TUP/UB));
   end
   Clock = Clock_S+PPD;
end
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function PPnodeS.m is used by PollingS.m function to clock the simulation run time and the communication traffic from the peripheral nodes to the central node.

```
function [Clock,B,P,SP,U,NE] = PPnodeS(Clock,ST,Pack,PPD,NE)

B=0;
P=0;
SP = 0;
U = 0;
TCPUD = Pack(4);
TDP = Pack(1);
TUP = Pack(3);
DB = Pack(7);
UB = Pack(8);

Clock = Clock +TCPUD;
if Clock < ST
   Clock_S = Clock+TUP;
   if Clock_S < ST
```

```
        B = UB;
        U = 1;
            Clock_S = Clock_S+TDP;
            if Clock_S < ST
                B = B+DB;
                P = 1;
            else
                B = B+DB-ceil((Clock_S-ST)/(TDP/DB));
            end
    else
        B = UB - ceil((Clock_S-ST)/(TUP/UB));
    end
    Clock = Clock_S+PPD;
end
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function PCnode2.m is used by both Polling2.m and PollingS.m functions to clock the simulation run time and the communication traffic from the central to the peripheral node.

```
function [Clock,B,NE] = PCnode2(Clock,S,Pack,PPD,NE)

B=0;
TCPUD = Pack(4);
TUP = Pack(3);
UB = Pack(8);

Clock = Clock+TCPUD;
if Clock < S
    Clock_S = Clock+TUP;
    if Clock_S > S
        B = ceil((Clock_S - S)/(TUP/UB));

    else
        B = UB;
        Clock_S = Clock_S+PPD;
    end
    Clock = Clock_S;
end
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

The function TokenH.m is called in the seastarsim.m function and is used to simulate the token ring with SRQ protocol.

```
% Function for <TokenH> uses the following input
% a. ST = Simulation Time
% b. SSpack(1) = TDP = Tx Time for a Data Packet
% c. SSpack(2) = TDSP = Tx Time for a Data Sub-packet
% d. SSpack(3) = TUP = Tx Time for a Utility packet
% e. SSpack(4) = TCPUD = Delay time for modem CPU
% f. SSpack(7) = DB = Number of Data bytes
% g. SSpack(8) = UB = Number of Utility bytes
% h. TDO = A array of Propagation delay between nodes in sequence 1 to N
% i. SSpack(9) = TTP = Tx time for a token ring
% k. SSpack(10) = RB = Number of Token Ring bytes.
%
% Output of <Token> will consist of the following
% a. TSDP = Successful number of data packet
% b. TSUP = Successful number of Utilities sign
% c. TSB = Successful number of byte Tx.
% d. TSPRT = Number of sub-packet re-transmitted.
% e. TFSP = Number of sub-packet failure with time of failures
% f. TFP = Number of packet failure with time of failures
% g. TFU = Number of failed utilities Tx failures with time log of failure
%
function [TSDP,TSTP,TSB,TSPRT,TFP,TFSP,TFU,TTO,CT] =
TokenH(ST,SSpack,TDO,EL)

Log = [];

%Ini Setup for Token function
time = 0; %Run time
sctime = 0; % Start cycle time
NE = 0;     % Running Number of Events
TSDP = 0;   % Number of successful Data packets transmitted
TSTP = 0;   % Number of successful Token packets transmitted
TSB = 0 ;   % Number of Bytes transmitted
TSPRT = 0;  % Number of sub-packet retransmitted
TFP = 0;    % Number of failed packet event
TFSP =0;    % Number of failed sub-packet
TFU = 0;    % Number of failed Utilities
TTO = 0;    % Number of time-outs
nCycle = 0; % Number of cycles
sCycle = 0;
avgCycletime = 0; % Average time per cycle
maxCycletime = 0; % max time per cycle
```

```
maxSRQ = 16;
maxRetry = 3;% Number of SRQ retries
N = size(TDO,2); %Number of Nodes
NCount = 2; % Counter for Node
TCPU = SSpack(4); %CPU Wake-up time
TDP = SSpack(1);  %Time for Tx of a data packet
TTok = SSpack(9); % Time for Tx of a token
TB = SSpack(10); % Number of Token Bytes
TSPB = SSpack(11); % Number of Byte per subpacket
Time_CP = TDO(2,:); % Matrix of time from central node to peripheral Node
state = 1;
TCompute = .3; %Central node CPU Computational time.
TO = 1; %additional time-out wait
ELU = EL(1); % Error Level of Utilities
ELP = EL(2); % Error Level of Data Packet
DropPackFlag = 0; % flagged if there's a dropped packet

% Create a look-up matrix of transmission time for each node
% consisting of propagation delay, Token, Full Data Tx, Delays

TX_Table = zeros(4,N);
TX_Table(1,:) = max(TDO,[],1); % Time taken to propagate to next node
TX_Table(2,:) = TTok; % Time taken to transmit a token
TX_Table(3,2:N) = TDP;  % Time take to transmit a data packet
TX_Table(4,:) = TCPU; % Time take for CPU delay
TX_Table(4,1) = TX_Table(4,1) + TCompute; % additional computational time for
central node

Tx_time = sum(TX_Table,1);

Time_CP = Time_CP + TTok+TCPU; %Time taken to Tx a token from Central node to
Peripheral node

% Create an array of number of SRQ Required (if any) and set as zero

N_SRQ = zeros(1,N);

% Create a matrix of flags to indicate error occurrances during Token and Data Tx

while time <= ST
   switch state

      case 1
         [time,B,NE] = TCRing(time,ST,SSpack,Time_CP,Log,NE,NCount);
         if AnyError(ELU) == true;
```

```
        state = 2;
    else
        state = 3;
        TSTP = TSTP + 1;
        TSB = TSB + B;
    end

case 2
    time = time + TDO(2,NCount) + TO;
    N_SRQ(NCount) = maxSRQ;
    NCount = NCount+1;
    if NCount > N
        state = 4; %Last node reached, go to error correction
        NCount = 2;
    else
        state = 1; %Sent Token from CNode to next PNode
    end

case 3
    [time,B,P,SP,NE] = TPHRing(time,ST,SSpack,Tx_time,Log,NE,NCount);
    if time > ST
        TSDP = TSDP + P;
        TSB = TSB + B;
    else
        if AnyError(ELP) == true %On Data Tx time
            N_SRQ(NCount) = ceil(random('unif',1,maxSRQ));
            TSDP = TSDP - 1;
            TSB = TSB - (N_SRQ(NCount)*TSPB);
        end
        TSDP = TSDP + P;
        TSB = TSB + B;
        NCount = NCount + 1;
        if NCount > N
            state = 4; %Last node reached, go to error correction
            NCount = 2;
        else
            if AnyError(ELU) == true % On Token tx time
                state = 2;
                TSB = TSB - TB;
            else
                TSTP = TSTP + 1;
                state = 3;
            end
        end
    end
```

```
        case 4
            NSRQ = N_SRQ(NCount);
            if NSRQ == 0 % No SRQ Required
                NCount = NCount + 1;
                if NCount > N
                    state = 5; % Go to restart
                else
                    state = 4;
                end
            else
                time = time + TCPU;
                [time,TOFlag,B,SP] =
ARQ(SSpack,TDO(2,NCount),NSRQ,maxRetry,time,EL);
                if TOFlag ~= 0
                    TFP = TFP + 1;
                    TTO = TTO + 1;
                    DropPackFlag = 1;
                else
                    TSDP = TSDP + 1;
                    TSB = TSB + B;
                    TSPRT = TSPRT + SP;
                end
                NCount = NCount + 1;
                if NCount > N
                    state = 5; %Go to restart
                else
                    state = 4;
                end
            end

        case 5
            NCount = 2;
            nCycle = nCycle + 1;
            ctime = time - sctime;
            sctime = time; % start a new cycle
            if ctime > maxCycletime
                maxCycletime = ctime;
            end
            if DropPackFlag ~= 1
                sCycle = sCycle +1;
            end
            avgCycletime = (time/nCycle);
            DropPackFlag = 0;
            N_SRQ = zeros(1,N);
```

```
        state = 1;
    end
end

CT = [avgCycletime,maxCycletime,nCycle,sCycle];
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function TokenS.m is called in the seastarsim.m function and is to simulate the token ring without SRQ.

```
% Function for <TokenS> uses the following input
% a. ST = Simulation Time
% b. SSpack(1) = TDP = Tx Time for a Data Packet
% c. SSpack(2) = TDSP = Tx Time for a Data Sub-packet
% d. SSpack(3) = TUP = Tx Time for a Utility packet
% e. SSpack(4) = TCPUD = Delay time for modem CPU
% f. SSpack(7) = DB = Number of Data bytes
% g. SSpack(8) = UB = Number of Utility bytes
% h. TDO = an array of Propagation delay between nodes in sequence 1 to N
% i. SSpack(9) = TTP = Tx time for a token ring
% k. SSpack(10) = RB = Number of Token Ring bytes.
%
% Output of <Token> will consist of the following:
% a. TSDP = Successful number of data packets
% b. TSUP = Successful number of Utilities sign
% c. TSB = Successful number of byte Tx.
% d. TSPRT = Number of sub-packets re-transmitted.
% e. TFSP = Number of sub-packets failure. with time of failure
% f. TFP = Number of packets failure with time of failure
% g. TFU = Number of failed utility Tx failures with time log of failure
%
function [TSDP,TSTP,TSB,TFP,TFU,CT] = TokenS(ST,SSpack,TDO,EL)

%Ini Setup for Token function
time = 0; %Run time
sctime = 0; % Start cycle time
TErrorFlag = 0; % Token Error Flag
NE = 0;     % Running Number of Events
TSDP = 0;   % Number of successful Data packets transmitted
TSTP = 0;   % Number of successful Token packets transmitted
TSB = 0 ;   % Number of Byte transmitted
TFP = 0;    % Number of failed packets event
TFU = 0;    % Number of failed Utility packet
nCycle = 0; % Number of cycle
```

```
sCycle = 0; % Number of Successful Cycle
avgCycletime = 0; % Average time per cycle
maxCycletime = 0; % max time per cycle
N = size(TDO,2); %Number of Nodes
NCount = 2; % Counter for Node
TCPU = SSpack(4); %CPU Wake up time
TDP = SSpack(1);  %Time for Tx a data packets
TTok = SSpack(9); % Time for Tx a token
TB = SSpack(10); % Number of Token Bytes
TDPB = SSpack(7); % Number of Bytes per subpackets
Time_CP = TDO(2,:); % Matrix of time from central node to Peripheral Node
state = 1;
TCompute = .3; %Central node CPU Computational time.
TO = 1; %additional time-out wait
ELU = EL(1); % Error Level of Utilities
ELP = EL(2); % Error Level of Data Packet
DropPackFlag = 0; % flaged if there's a dropped packet

% Create a look up matrix of transmission time for each node
% consisting of propagation delay, Token, Full Data Tx, Delays

TX_Table = zeros(4,N);
TX_Table(1,:) = max(TDO,[],1); % Time taken to propagate to next node
TX_Table(2,:) = TTok; % Time taken to transmit a token
TX_Table(3,2:N) = TDP;  % Time take to transmit a data packets
TX_Table(4,:) = TCPU; % Time take for CPU delay
TX_Table(4,1) = TX_Table(4,1) + TCompute; % additional computational time for
central node

Tx_time = sum(TX_Table,1);

Time_CP = Time_CP + TTok+TCPU; %Time taken to Tx a token from Central node to
Peripheral node

% Create an array of number of SQR Required (if any) and set as zero


% Create a matrix of flag to indicate error occurances during Token and Data Tx

while time <= ST
   switch state

      case 1
         [time,B,NE] = TCRing(time,ST,SSpack,Time_CP,Log,NE,NCount);
         if AnyError(ELU) == true;
```

```
        state = 2;
        TFU = TFU + 1;
        DropPackFlag = 1;
    else
        state = 3;
        TSTP = TSTP + 1;
        TSB = TSB + B;
    end

case 2
    time = time + TDO(2,NCount) + TO;
    TFP = TFP + 1;
    NCount = NCount+1;
    if NCount > N
        state = 4; %Last node reached
        NCount = 2;
    else
        state = 1; %Sent Token from CNode to next PNode
    end

case 3
    [time,B,P,SP,NE] = TPHRing(time,ST,SSpack,Tx_time,Log,NE,NCount);
    if time > ST
        TSDP = TSDP + P;
        TSB = TSB + B;
    else
        if AnyError(ELP) == true %On Data Tx time
            TSDP = TSDP - 1;
            TSB = TSB - TDPB;
            DropPackFlag = 1;
        end
        TSDP = TSDP + P;
        TSB = TSB + B;
        NCount = NCount+1;
        if AnyError(ELU) == true % On Token Tx time
            TFU = TFU+1;
            DropPackFlag = 1;
            if NCount > N
                state = 4;
                TErrorFlag = 1;
            else
                state = 2;
            end
            TSB = TSB - TB;
        else
```

```
            if NCount > N
               state = 4;
            else
               state = 3;
               TSTP = TSTP +1;
            end
         end
      end

   case 4
      NCount = 2;
      nCycle = nCycle + 1;
      ctime = time - sctime;
      sctime = time; % start a new cycle
      if ctime > maxCycletime
         maxCycletime = ctime;
      end
      if DropPackFlag ~= 1
            sCycle = sCycle+1;
      end
      avgCycletime = (time/nCycle);
      DropPackFlag = 0;
      if TErrorFlag == 1
         state = 2;
         TErrorFlag = 0;
      else
         state = 3;
      end
   end
end

CT = [avgCycletime,maxCycletime,nCycle,sCycle];
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

This function TPHRing.m is used by both the TokenH.m and TokenS.m function to clock the simulation time and the communication traffic between each peripheral nodes.

```
function [Clock,B,P,SP,NE] = TPHRing(Clock,ST,Pack,Tx,Log,NE,NC)

B=0;
P=0;
SP = 0;
TCPUD = Pack(4);
TDP = Pack(1);
```

```
TTP = Pack(9);
DB = Pack(7);
RB = Pack(10);
state = 1;
Endstate = 0;

Clock_S = Clock +TCPUD;
while Endstate == 0
   switch state

      case 1
         Clock_S = Clock + TDP;
         if Clock_S > ST
            B = B+floor((ST - Clock)/(TDP/DB));
            state = 4; %End State
         else
            B = B+DB;
            P = P+1;
            state = 2;
         end

      case 2
         Clock_S = Clock_S+TTP;
         if Clock_S > ST
            B = floor((ST-Clock)/(TTP/RB));
            state = 4;
         else
            B = B+RB;
            state = 3;
         end

      case 3
         Clock = Clock+Tx(NC)-TCPUD;
         Endstate = 1;

      case 4
         Clock = Clock_S;
         Endstate = 1;
   end
end
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function TCRing.m is used by both the TokenH.m and TokenS.m function to clock the simulation time and the communication traffic between the central node and the peripheral nodes.

```
function [Clock,B,NE] = TCRing(Clock,S,Pack,Tk,Log,NE,NC)

B=0;
TCPUD = Pack(4);
TTP = Pack(9);
RB = Pack(10);

Clock = Clock+TCPUD;
if Clock < S
   Clock_S = Clock+TTP;
   if Clock_S > S
      B = floor((S - Clock)/(TTP/RB));
   else
      B = RB;
   end
   Clock = Clock + Tk(NC);
end
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function ARQ.m is used by the function TokenH.m to generate SRQ events when the token returns to the central node after each cycle.

```
%This function take in the following
%Spack contains the necessary information.
%SPack(2) = TimeDSP = Tx time per data sub-packets
%Spack(3) = TimeUtil = Tx time per utilities packets
%Spack(4) = TCPU = CPU wake-up and computation time
%Spack(10) = UB = No. of Utilities bytes
%Spack(11) = SB = No. of Sub-packets bytes
%TimeDelay = Propagation delay between the two nodes
%NSRQ = Number of SRQ packets required
%time = run time clock
%
% The function returns the following
%TOFlag = Time-Out Flag
%B = Number of byte successfully transmitted
%SP = Number of sub-packet successfully transmitted
function [time,TOFlag,B,SP] = ARQ(Spack,TimeDelay,NSRQ,MaxTry,time,EL)

TimeDSP = Spack(2);
```

```
TimeUtil = Spack(3);
TCPU = Spack(4);
UB = Spack(10);
SB = Spack(11);
Try = 1;
TOFlag = 0;
Endflag = 0;
B = 0;
SP = 0;
state = 1;
ELU = EL(1); % Error Level of Utilities
ELP = EL(2); % Error Level of Data Packet

while Endflag < 1
   switch state

      case 1
         %Send SRQ to Pnode
         time = time + TimeUtil + TimeDelay;
         if AnyError(ELU) == true;
            time = time + TimeDelay + TCPU + (TimeDSP*NSRQ); %Wait time for data
return
            Try = Try + 1;
            if Try > MaxTry
               state = 4;
            else
               state = 1;
            end
         else
            B = B + UB;
            state = 2;
         end

      case 2 %Send Data to Cnode
         time = time + (TimeDSP*NSRQ) + TCPU + TimeDelay;
         if AnyError(ELP) == true;
            Try = Try + 1;
            if Try > MaxTry
               state = 4;
            else
               PNSRQ = NSRQ;
               NSRQ = ceil(random('unif',1,NSRQ));
               B = B + (PNSRQ - NSRQ)*SB;
               SP = SP + PNSRQ - NSRQ;
               state = 1;
```

```
            end
        else
            B = B + (NSRQ*SB);
            SP = SP + NSRQ;
            state = 3;
        end

    case 3
        Endflag = 1;

    case 4
        TOFlag = 1;
        Endflag = 1;
  end
end
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function AnyError.m will return a logic True when the random uniform generator exceed the EL level. It is used to generate the error inthe program.

```
function [AE] = AnyError(EL)

AE = EL > random('unif',0,1);
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

The function of seastarsim.m is to run the same set of input through the four different protocols and collect the result for analysis

```
function [Result] = seastarsim(SR,STH,NoNode,NGPos,EL)

[SSpack] = iniSeaStar;
SSpack(6) = NoNode;
NPos = NGPos;

% Call a function <GridNode> with a given matrix input of the
% node position and convert it into a grid of 1000m by 1000m.
% Output of the function <MapNode> will consist of the grid position of
% the node and the distance between all the nodes.

NGrid = GridNode(SSpack(6),NPos);
```

```
% Call a function <Proptime> with input consisiting of output of MapNode
% and sound speed in order to calculate the propagation delay between the
% nodes.
% Output of the function <Proptime> will consist of the grid position of
% the node and the propagation time delay between the nodes.

TPropDelay = PropTime(SSpack(6),SSpack(5),NGrid);

% Call function <TkRing> with input of TProgDelay and Number of nodes and
% output the order that will minimize the delay.
[TRorder,TRDelayOrder] = TkRing(TPropDelay,SSpack(6));

%Obtain the Propagation delay for Polling
PPropDelay = TPropDelay(3,:);

% Set simulation time <SimTime> in Hours
SimTimeHours = STH;

%Convert SimTime to seconds
SimTime = SimTimeHours*3600;

% Load number of simulation
Simrun = SR;
Prun = zeros(Simrun,12);
PSrun = zeros(Simrun,10);
Trun = zeros(Simrun,12);
TSrun = zeros(Simrun,9);

% Run simulation for <Polling> for the above initialization and noise.

for i = 1:Simrun
   [PsSDP,PsSUP,PsSB,PsFP,PsFU,PsTO,CT] = PollingS(SimTime,SSpack,...
      PPropDelay,EL);

   PSrun(i,:) = [PsSDP,PsSUP,PsSB,PsFP,PsFU,PsTO,CT];

   [PSDP,PSUP,PSB,PSPRT,PFP,PFSP,PFU,PTO,CT] = Polling2(SimTime,SSpack,...
      PPropDelay,EL);

   Prun(i,:) = [PSDP,PSUP,PSB,PSPRT,PFP,PFSP,PFU,PTO,CT];


% Run simulation for <TokenH> for the above initialization and noise.

[TSDP,TSTP,TSB,TSPRT,TFP,TFSP,TFU,TTO,CT] =
```

```
TokenH(SimTime,SSpack,TRDelayOrder,EL);

 Trun(i,:) = [TSDP,TSTP,TSB,TSPRT,TFP,TFSP,TFU,TTO,CT];

% Run Simulation for <TokenS> as simple Token ring format
 [STSDP,STSTP,STSB,STFP,STFU,CT] =
TokenS(SimTime,SSpack,TRDelayOrder,EL);

 TSrun(i,:) = [STSDP,STSTP,STSB,STFP,STFU,CT];

end

% Generate a summary.

Result = zeros(4,14);

% Polling Simple Run
SumPSrun = sum(PSrun,1);
MaxPSrun = max(PSrun,[],1);
MinPSrun = min(PSrun,[],1);
PSAvgTimePerCycle = SumPSrun(7)/Simrun;
PSMinAvgTimePerCycle = MinPSrun(7);
PSMaxAvgTimePerCycle = MaxPSrun(7);
PSAvgMaxTimePerCycle = SumPSrun(8)/Simrun;
PSMinAMaxTimePerCycle = MinPSrun(8);
PSMaxAMaxTimePerCycle = MaxPSrun(8);

ThroughPut_PSPacket = (SumPSrun(1)*3600/(SimTime*Simrun))*24; %packets/day
ThroughPut_PSData = (SSpack(12)*SumPSrun(1))/(SimTime*Simrun); %byte/sec
ThroughPut_PSUtil = (SSpack(8)*SumPSrun(2))/(SimTime*Simrun); %byte/sec
Total_PSByte = SumPSrun(3);

Dropped_PSData = SSpack(7)*SumPSrun(4)/Simrun;
Dropped_PSPData = (SumPSrun(4)*3600/(SimTime*Simrun))*24;

Channel_PSData = (ThroughPut_PSData/SSpack(13))*100; % % of Channel Utilization

nCycle_PS = (SumPSrun(9)*3600/(SimTime*Simrun))*24; %Total Cycles/day
sCycle_PS = (SumPSrun(10)*3600/(SimTime*Simrun))*24; % Total Successful
Cycles/day


Result(1,:) =
[PSAvgTimePerCycle,PSMinAvgTimePerCycle,PSMaxAvgTimePerCycle,...
   PSAvgMaxTimePerCycle,PSMinAMaxTimePerCycle,PSMaxAMaxTimePerCycle,...
```

```
    ThroughPut_PSPacket,ThroughPut_PSData,ThroughPut_PSUtil,Dropped_PSPData,...
    Total_PSByte,Channel_PSData,nCycle_PS,sCycle_PS];

% Polling with SRQ Run
SumPrun = sum(Prun,1);
MaxPrun = max(Prun,[],1);
MinPrun = min(Prun,[],1);
PAvgTimePerCycle = SumPrun(9)/Simrun;
PMinAvgTimePerCycle = MinPrun(9);
PMaxAvgTimePerCycle = MaxPrun(9);
PAvgMaxTimePerCycle = SumPrun(10)/Simrun;
PMinAMaxTimePerCycle = MinPrun(10);
PMaxAMaxTimePerCycle = MaxPrun(10);

ThroughPut_PPacket = (SumPrun(1)*3600/(SimTime*Simrun))*24; %packets/day
ThroughPut_PData = (SSpack(12)*SumPrun(1))/(SimTime*Simrun);
ThroughPut_PUtil = (SSpack(8)*SumPrun(2))/(SimTime*Simrun);
Total_PByte = SumPrun(3);

Dropped_PData = SSpack(7)*SumPrun(8)/Simrun;
Dropped_PPData =(SumPrun(8)*3600/(SimTime*Simrun))*24;

Channel_PData = (ThroughPut_PData/SSpack(13))*100; % % of Channel Utilization

nCycle_P = (SumPrun(11)*3600/(SimTime*Simrun))*24; %Total Cycles/day
sCycle_P = (SumPrun(12)*3600/(SimTime*Simrun))*24; % Total Successful Cycles/day

Result(2,:) = [PAvgTimePerCycle,PMinAvgTimePerCycle,PMaxAvgTimePerCycle,...
    PAvgMaxTimePerCycle,PMinAMaxTimePerCycle,PMaxAMaxTimePerCycle,...
    ThroughPut_PPacket,ThroughPut_PData,ThroughPut_PUtil,Dropped_PPData,...
    Total_PByte,Channel_PData,nCycle_P,sCycle_P];

% Token Simple run
SumSTrun = sum(TSrun,1);
MaxSTrun = max(TSrun,[],1);
MinSTrun = min(TSrun,[],1);
STAvgTimePerCycle = SumSTrun(6)/Simrun;
STMinAvgTimePerCycle = MinSTrun(6);
STMaxAvgTimePerCycle = MaxSTrun(6);
STAvgMaxTimePerCycle = SumSTrun(7)/Simrun;
STMinAMaxTimePerCycle = MinSTrun(7);
STMaxAMaxTimePerCycle = MaxSTrun(7);

ThroughPut_STPacket = (SumSTrun(1)*3600/(SimTime*Simrun))*24; %packets/day
ThroughPut_STData = (SSpack(12)*SumSTrun(1))/(SimTime*Simrun);
```

```
ThroughPut_STUtil = (SSpack(8)*SumSTrun(2))/(SimTime*Simrun);
Total_STByte = SumSTrun(3);

Dropped_STData = SSpack(7)*SumSTrun(4)/Simrun;
Dropped_STPData = (SumSTrun(4)*3600/(SimTime*Simrun))*24;

Channel_STData = (ThroughPut_STData/SSpack(13))*100; % % of Channel Utilization

nCycle_ST = (SumSTrun(8)*3600/(SimTime*Simrun))*24; %Total Cycles/day
sCycle_ST = (SumSTrun(9)*3600/(SimTime*Simrun))*24; % Total Successful
Cycles/day

Result(3,:) =
[STAvgTimePerCycle,STMinAvgTimePerCycle,STMaxAvgTimePerCycle,...
   STAvgMaxTimePerCycle,STMinAMaxTimePerCycle,STMaxAMaxTimePerCycle,...
   ThroughPut_STPacket,ThroughPut_STData,ThroughPut_STUtil,Dropped_STPData,...
   Total_STByte,Channel_STData,nCycle_ST,sCycle_ST];

% Token with SRQ run
SumTrun = sum(Trun,1);
MaxTrun = max(Trun,[],1);
MinTrun = min(Trun,[],1);
TAvgTimePerCycle = SumTrun(9)/Simrun;
TMinAvgTimePerCycle = MinTrun(9);
TMaxAvgTimePerCycle = MaxTrun(9);
TAvgMaxTimePerCycle = SumTrun(10)/Simrun;
TMinAMaxTimePerCycle = MinTrun(10);
TMaxAMaxTimePerCycle = MaxTrun(10);

ThroughPut_TPacket = (SumTrun(1)*3600/(SimTime*Simrun))*24; %packets/day
ThroughPut_TData = (SSpack(12)*SumTrun(1))/(SimTime*Simrun);
ThroughPut_TUtil = (SSpack(8)*SumTrun(2))/(SimTime*Simrun);
Total_TByte = SumTrun(3);

Dropped_TData = SSpack(7)*SumTrun(8)/Simrun;
Dropped_TPData = (SumTrun(8)*3600/(SimTime*Simrun))*24;

Channel_TData = (ThroughPut_TData/SSpack(13))*100; % % of Channel Utilization

nCycle_T = (SumTrun(11)*3600/(SimTime*Simrun))*24; %Total Cycles/day
sCycle_T = (SumTrun(12)*3600/(SimTime*Simrun))*24; % Total Success Cycles/day

Result(4,:) = [TAvgTimePerCycle,TMinAvgTimePerCycle,TMaxAvgTimePerCycle,...
   TAvgMaxTimePerCycle,TMinAMaxTimePerCycle,TMaxAMaxTimePerCycle,...
   ThroughPut_TPacket,ThroughPut_TData,ThroughPut_TUtil,Dropped_TPData,...
```

```
Total_TByte,Channel_TData,nCycle_T,sCycle_T];
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This function iniGrid.m is called in the SeaStarWNodesSim main prgram to initialize the grid layout of nodes.

```
function [Linear,Circle] = iniGrid

Linear = cell(7,1);
Circle = cell(7,1);

Linear{1} = [0,250,500;0,0,0];
Linear{2} = [0,167,333,500;0,0,0,0];
Linear{3} = [0,125,250,375,500;0,0,0,0,0];
Linear{4} = [0,100,200,300,400,500;0,0,0,0,0,0];
Linear{5} = [0,-500,-333,-167,167,333,500;0,0,0,0,0,0,0];
Linear{6} = [0,-500,-333,-167,125,250,375,500;0,0,0,0,0,0,0,0];
Linear{7} = [0,-500,-375,-250,-125,125,250,375,500;0,0,0,0,0,0,0,0,0];

Circle{1} = [0,217,-217;0,450,450];
Circle{2} = [0,0,250,-250;0,289,-144,-144];
Circle{3} = [0,0,354,0,-354;0,354,0,-354,0];
Circle{4} = [0,0,405,250,-250,-405;0,425,131,-344,-344,131];
Circle{5} = [0,250,500,250,-250,-500,-250;0,433,0,-433,-433,0,433];
Circle{6} = [0,0,391,487,217,-217,-487,-391;0,500,312,-111,-450,-450,-111,312];
Circle{7} = [0,0,354,500,354,0,-354,-500,-354;0,500,354,0,-354,-500,-354,0,354];
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This is a one of two main program name SeaStarWNodesSim to compute collect the data for 3 to 9 number of nodes.

```
% This program gives seastarsim.m two different geometrical layout and a set of number
of nodes to simulate

clear

SimTimeHours = 6;
Simrun = 150;
ErrorL = [0.0005,0.2];

[Lin,Cir] = iniGrid;

Count = size(Lin,1);
```

```
CPS = zeros(Count,14);
CPH = zeros(Count,14);
CTS = zeros(Count,14);
CTH = zeros(Count,14);
LPS = zeros(Count,14);
LPH = zeros(Count,14);
LTS = zeros(Count,14);
LTH = zeros(Count,14);


for a = 1:Count
   xy = Cir{a};
   N = size(xy,2);
   [Return] = seastarsim(Simrun,SimTimeHours,N,xy,ErrorL);
   CPS(a,:) = Return(1,:);
   CPH(a,:) = Return(2,:);
   CTS(a,:) = Return(3,:);
   CTH(a,:) = Return(4,:);
   xyl = Lin{a};
   NL = size(xyl,2);
   [Return] = seastarsim(Simrun,SimTimeHours,NL,xyl,ErrorL);
   LPS(a,:) = Return(1,:);
   LPH(a,:) = Return(2,:);
   LTS(a,:) = Return(3,:);
   LTH(a,:) = Return(4,:);

end

NN = 3:9;

figure(1)
plot(NN,CPS(:,1),'co-');
hold on
plot(NN,CPH(:,1),'bx-');
plot(NN,CTS(:,1),'g+-');
plot(NN,CTH(:,1),'m*-');
plot(NN,LPS(:,1),'co-.');
plot(NN,LPH(:,1),'bx-.');
plot(NN,LTS(:,1),'g+-.');
plot(NN,LTH(:,1),'m*-.');
hold off
xlabel('Number of Nodes');ylabel('Time (sec)');title('Average Time Per Cycle vs Number
of Nodes @ EL = 0.2');
legend('C. Polling','C. Polling with SRQ','C. Token',...
```

```
    'C. Token with SRQ','L. Polling','L. Polling with SRQ',...
    'L. Token','L. Token with SRQ','Location','NorthWest');

figure(2)
plot(NN,CPS(:,4),'co-');
hold on
plot(NN,CPH(:,4),'bx-');
plot(NN,CTS(:,4),'g+-');
plot(NN,CTH(:,4),'m*-');
plot(NN,LPS(:,4),'co-.');
plot(NN,LPH(:,4),'bx-.');
plot(NN,LTS(:,4),'g+-.');
plot(NN,LTH(:,4),'m*-.');
hold off
xlabel('Number of Nodes');ylabel('Time (sec)');title('Max Time Per Cycle vs Number of
Nodes @ EL = 0.2');
legend('C. Polling','C. Polling with SRQ','C. Token',...
    'C. Token with SRQ','L. Polling','L. Polling with SRQ',...
    'L. Token','L. Token with SRQ','Location','NorthWest');

figure(3)
plot(NN,CPS(:,7),'co-');
hold on
plot(NN,CPH(:,7),'bx-');
plot(NN,CTS(:,7),'g+-');
plot(NN,CTH(:,7),'m*-');
plot(NN,LPS(:,7),'co-.');
plot(NN,LPH(:,7),'bx-.');
plot(NN,LTS(:,7),'g+-.');
plot(NN,LTH(:,7),'m*-.');
hold off
xlabel('Number of Nodes');ylabel('Number of Packet (per day)');title('Throughput for
Packet vs Number of Nodes @ EL = 0.2');
legend('C. Polling','C. Polling with SRQ','C. Token',...
    'C. Token with SRQ','L. Polling','L. Polling with SRQ',...
    'L. Token','L. Token with SRQ','Location','Best');

figure(4)
plot(NN,CPS(:,8),'co-');
hold on
plot(NN,CPH(:,8),'bx-');
plot(NN,CTS(:,8),'g+-');
plot(NN,CTH(:,8),'m*-');
plot(NN,LPS(:,8),'co-.');
plot(NN,LPH(:,8),'bx-.');
```

```
plot(NN,LTS(:,8),'g+-.');
plot(NN,LTH(:,8),'m*-.');
hold off
xlabel('Number of Nodes');ylabel('Number of Data Byte Per Sec');title('Throughput Data
vs Number of Nodes @ EL = 0.2');
legend('C. Polling','C. Polling with SRQ','C. Token',...
    'C. Token with SRQ','L. Polling','L. Polling with SRQ',...
    'L. Token','L. Token with SRQ','Location','Best');

figure(5)
plot(NN,CPS(:,10),'co-');
hold on
plot(NN,CPH(:,10),'bx-');
plot(NN,CTS(:,10),'g+-');
plot(NN,CTH(:,10),'m*-');
plot(NN,LPS(:,10),'co-.');
plot(NN,LPH(:,10),'bx-.');
plot(NN,LTS(:,10),'g+-.');
plot(NN,LTH(:,10),'m*-.');
hold off
xlabel('Number of Nodes');ylabel('Number of Dropped packet (per day)');title('Dropped
Data vs Number of Nodes @ EL = 0.2');
legend('C. Polling','C. Polling with SRQ','C. Token',...
    'C. Token with SRQ','L. Polling','L. Polling with SRQ',...
    'L. Token','L. Token with SRQ','Location','East');



figure(6)
plot(NN,CPS(:,12),'co-');
hold on
plot(NN,CPH(:,12),'bx-');
plot(NN,CTS(:,12),'g+-');
plot(NN,CTH(:,12),'m*-');
plot(NN,LPS(:,12),'co-.');
plot(NN,LPH(:,12),'bx-.');
plot(NN,LTS(:,12),'g+-.');
plot(NN,LTH(:,12),'m*-.');
hold off
xlabel('Number of Nodes');ylabel('Channel Utilization Rate (%)');
title('Channel Utilization Rate vs Number of Nodes @ EL = 0.2');
legend('C. Polling','C. Polling with SRQ','C. Token',...
    'C. Token with SRQ','L. Polling','L. Polling with SRQ',...
    'L. Token','L. Token with SRQ','Location','Best');
```

```
figure(7)
plot(NN,CPS(:,14),'co-');
hold on
plot(NN,CPH(:,14),'bx-');
plot(NN,CTS(:,14),'g+-');
plot(NN,CTH(:,14),'m*-');
plot(NN,LPS(:,14),'co-.');
plot(NN,LPH(:,14),'bx-.');
plot(NN,LTS(:,14),'g+-.');
plot(NN,LTH(:,14),'m*-.');
hold off
xlabel('Number of Nodes');ylabel('Cycle (per day');
title('Error Free Cycle vs Number of Nodes @ EL = 0.2');
legend('C. Polling','C. Polling with SRQ','C. Token',...
   'C. Token with SRQ','L. Polling','L. Polling with SRQ',...
   'L. Token','L. Token with SRQ','Location','NorthEast');
```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

This is the second of two main program named SeaStarWError which is used to measure the varying degree of error rate for a 7 nodes network.

```
% This program gives seastarsim.m a error level set to run
clear

SimTimeHours = 6;
Simrun = 100;
ELP = [0,0.05,0.1,0.2,0.4,0.6,0.8]; %Error Lvl of Data Packet
ELU = ELP./400; %Error lvl of Util packet

% 6 peripheral nodes circle
xy = [0,0,0,433,-433,433,-433;
   0,500,-500,250,250,-250,-250];
N = size(xy,2);

% Other ini
b = size(ELP,2);
CPS = zeros(b,14);
CPH = zeros(b,14);
CTS = zeros(b,14);
CTH = zeros(b,14);


for a = 1:b
```

```
    ErrorL = [ELU(a),ELP(a)];
    [Return] = seastarsim(Simrun,SimTimeHours,N,xy,ErrorL);
    CPS(a,:) = Return(1,:);
    CPH(a,:) = Return(2,:);
    CTS(a,:) = Return(3,:);
    CTH(a,:) = Return(4,:);
end

figure(1)
plot(ELP,CPS(:,1),'co-');
hold on
plot(ELP,CPH(:,1),'bx-');
plot(ELP,CTS(:,1),'g+-');
plot(ELP,CTH(:,1),'m*-');
hold off
xlabel('Error Level');ylabel('Time (sec)');title('Average Time Per Cycle vs Error Rate');
legend('Polling','Polling with SRQ','Token',...
    'Token with SRQ','Location','NorthWest');
xlim([0,0.8]);

figure(2)
plot(ELP,CPS(:,4),'co-');
hold on
plot(ELP,CPH(:,4),'bx-');
plot(ELP,CTS(:,4),'g+-');
plot(ELP,CTH(:,4),'m*-');
hold off
xlabel('Error Level');ylabel('Time (sec)');title('Max Time Per Cycle vs Error Rate');
legend('Polling','Polling with SRQ','Token',...
    'Token with SRQ','Location','NorthWest');
xlim([0,0.8]);

figure(3)
plot(ELP,CPS(:,7),'co-');
hold on
plot(ELP,CPH(:,7),'bx-');
plot(ELP,CTS(:,7),'g+-');
plot(ELP,CTH(:,7),'m*-');
hold off
xlabel('Error Level');ylabel('Number of Packet (per day)');title('Throughput for Packet vs
Error Rate');
legend('Polling','Polling with SRQ','Token',...
    'Token with SRQ','Location','NorthEast');
xlim([0,0.8]);
```

```
figure(4)
plot(ELP,CPS(:,8),'co-');
hold on
plot(ELP,CPH(:,8),'bx-');
plot(ELP,CTS(:,8),'g+-');
plot(ELP,CTH(:,8),'m*-');
hold off
xlabel('Error Level');ylabel('Number of Data Byte Per Sec');title('Throughput Data vs
Error Rate');
legend('Polling','Polling with SRQ','Token',...
   'Token with SRQ','Location','NorthEast');
xlim([0,0.8]);

figure(5)
plot(ELP,CPS(:,10),'co-');
hold on
plot(ELP,CPH(:,10),'bx-');
plot(ELP,CTS(:,10),'g+-');
plot(ELP,CTH(:,10),'m*-');
hold off
xlabel('Error Level');ylabel('Number of Dropped packet (per day)');title('Dropped Data vs
Error Rate');
legend('Polling','Polling with SRQ','Token',...
   'Token with SRQ','Location','NorthWest');
xlim([0,0.8]);

figure(6)
plot(ELP,CPS(:,12),'co-');
hold on
plot(ELP,CPH(:,12),'bx-');
plot(ELP,CTS(:,12),'g+-');
plot(ELP,CTH(:,12),'m*-');
hold off
xlabel('Error Level');ylabel('Channel Utilization Rate (%)');
title('Channel Utilization Rate vs Error Rate');
legend('Polling','Polling with SRQ','Token',...
   'Token with SRQ','Location','NorthEast');
xlim([0,0.8]);%ylim([0,100]);

figure(7)
plot(ELP,CPS(:,14),'co-');
hold on
plot(ELP,CPH(:,14),'bx-');
plot(ELP,CTS(:,14),'g+-');
plot(ELP,CTH(:,14),'m*-');
```

```
hold off
xlabel('Error Level');ylabel('Cycle (per day)');
title('Error Free Cycle vs Error Rate');
legend('Polling','Polling with SRQ','Token',...
    'Token with SRQ','Location','NorthEast');
xlim([0,0.8]);
```

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES


[1] B. E. A. Kerstens, "A study of the Seastar underwater acoustic Local Area Network Concept," M.S. thesis, Naval Postgraduate School, Monterey, CA, December 2007.

[2] Teledyne Benthos Staff, *Acoustic Telemetry Modem User's Manual*, Teledyne Benthos, P/N 003452, rev. F, March 2006.

[3] W. Jenkins, "Time/frequency relationships for an FFT-based acoustic modem," M.S. thesis, Naval Postgraduate School, Monterey, CA, June 2010.

[4] W. H. Thorp, "Analytic description of low frequency attenuation coefficient," *Journal of the Acoustical Society of America*, vol. 42, pp. 270–271, 1967.

[5] R. F. W. Coates, *Underwater Acoustic System*, NewYork: Halsted Press, 1986.

[6] R. Urick, *Principles of Underwater Sound*. 3rd ed. Los Altos, CA: Peninsula Publishing, 1983.

[7] A. L. Anderson and G. J. Gruber, "Ambient-noise measurements at 30, 90, and 150kHz in Five Ports," Journal of the Acoustical Society of America, vol. 49, no. 3, pp. 928–930, 1971.

[8] R. Urick, *Ambient noise at the sea*, Los Altos: Peninsula Publishing, 1986.

[9] J. R. Potter, T. W. Lim and M. Chitre, "Acoustic imaging & The natural soundscape in Singapore waters," Proceeding of Mindef-NUS joint seminar, pp. 141–147, 1997.

[10] H. Medwin, *Sounds in the sea: From ocean acoustics to acoustical Oceanography*, Cambridge University Press, 2005.

[11] J. G. Proakis, *Digital communications*, 4th ed., McGraw-Hill, 2001.

[12] J. T. Hanson, "Link budget analysis for undersea acoustic signaling," M.S. thesis, Naval Postgraduate School, Monterey, CA, June 2002.

[13] J. M. Kalscheuer, "A Selective Automatic Repeat Request Protocol for Undersea Acoustic Links," M.S. thesis, Naval Postgraduate School, Monterey, CA, June 2004.

[14] M. J. Hahn, "Undersea navigation via a distributed acoustic communication network," M.S. thesis, Naval Postgraduate School, Monterey, CA, June 2005.

[15] L.E. Kinsler, A.R. Frey, A.B. Coppens and J.V. Sanders, *Fundamentals of Acoustics*, 4th ed., John Wiley & Sons Inc, 2000.

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Joseph A. Rice
   Naval Postgraduate School
   Monterey, California

4. Professor Daphne Kapolka
   Naval Postgraduate School
   Monterey, California

5. Roald Otnes
   Norwegian Defense Research Establishment
   Horten, Norway

6. Chris Fletcher
   SPAWAR Systems Center Pacific
   San Diego, California

7. ENS William Jenkins
   Naval Postgraduate School
   Monterey, California

8. LT Pongsakorn Sommai
   Naval Postgraduate School
   Monterey, California

9. ME5 Goh Meng Chong
   Republic of Singapore Navy
   Singapore